PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-326078
(43)Date of publication of application : 08.12.1998

(51)Int.Cl. G09C 1/00
G09C 1/00
H04L 9/32

(21)Application number : 09-363804
(22)Date of filing : 27.11.1997

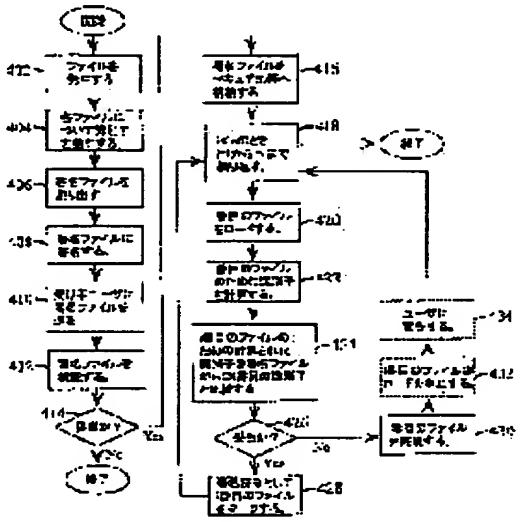
(71)Applicant : SUN MICROSYST INC
(72)Inventor : RENAUD BENJAMIN J
PAMPUCH JOHN C
HODGES WILSHER AVRIL E

(30)Priority

Priority number : 96 753716 Priority date : 27.11.1996 Priority country : US
97 780817 09.01.1997 US

(54) IMPLEMENTATION OF DIGITAL SIGN FOR DATA STREAM AND DATA ARCHIVES

(57)Abstract:
PROBLEM TO BE SOLVED: To guarantee and verify the genuineness of a data file by processing a sign file by using a computer system to judge the genuineness of the sign file.
SOLUTION: Identifiers are generated for respective data files (ST404). Then a sign file for listing or compiling the identifiers is generated (ST406). This sign file is electronically signed by using sign algorithm (ST408). The signed sign file is sent to a receiving user and shown or made usable (ST410). The receiving user verifies the genuineness of the signed sign file which is sent or made usable at the time of reception or access (ST412). Then the validity of the electronic sign is determined (ST414). Then the identifier from the sign file is stored (ST416).



LEGAL STATUS

[Date of request for examination] 27.01.1998
[Date of sending the examiner's decision of rejection] 19.08.2002
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection] 2002-22234

NTT- 1733

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平10－326078

(43)公開日　平成10年(1998)12月８日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０９Ｃ　1/00	６４０	Ｇ０９Ｃ　1/00	６４０Ｚ
			６４０Ａ
			６４０Ｂ
			６４０Ｄ
	６６０		６６０Ｄ

審査請求　有　　請求項の数21　ＯＬ　外国語出願　（全 53 頁）　最終頁に続く

(21)出願番号　特願平9－363804

(22)出願日　平成９年(1997)11月27日

(31)優先権主張番号　０８／７５３７１６

(32)優先日　1996年11月27日

(33)優先権主張国　米国（ＵＳ）

(31)優先権主張番号　０８／７８０８１７

(32)優先日　1997年１月９日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　595034134
サン・マイクロシステムズ・インコーポレイテッド
Ｓｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓ，　Ｉｎｃ．
アメリカ合衆国　カリフォルニア州　94303　パロ　アルト　サン　アントニオ　ロード　901

(72)発明者　ベンジャミン　ジェイ．　レナウド
アメリカ合衆国，　カリフォルニア州，　ウッドサイド，　チャップマン　ロード　152

(74)代理人　弁理士　長谷川　芳樹　（外５名）

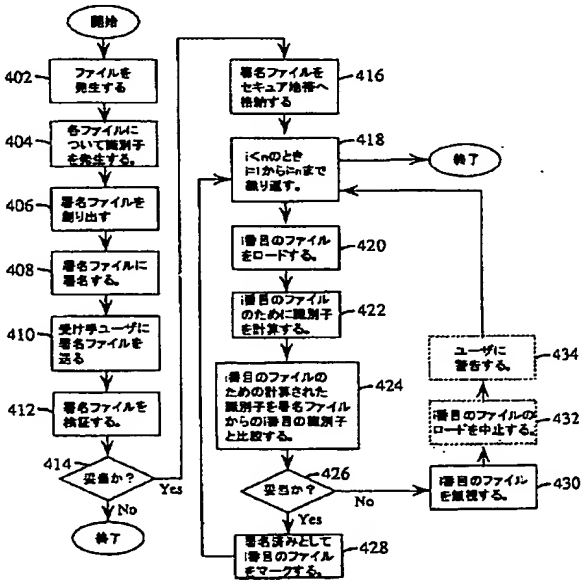
最終頁に続く

(54)【発明の名称】　データストリーム及びデータアーカイブのためのデジタル署名の実施

(57)【要約】

【課題】　ネットワークにわたって転送されるデータファイルの真正性を保証し検証するためのより効率的な方法、装置及び生産物を提供する。

【解決手段】　データの真正性を検証する方法は、識別子と、電子署名とデータファイルの識別子を持つ署名ファイルを持つようなデータファイルを少なくとも１つ提供する。電子署名はコンピュータシステムによって検証され、データファイル内の識別子はコンピュータシステムを使用して署名ファイル内の識別子と比較される。データファイルの識別子は、証明認証、サイト証明、ソフトウェア配布者の識別子、サイト名の少なくとも１つを持ち、データの真正性の検証には証明認証、前記サイト証明、前記ソフトウェア配布者の識別子、前記サイト名の少なくとも１つに関するセキュリティレベルの設定がある。



1

【特許請求の範囲】

【請求項１】　データの真正性を検証するための、コンピュータで実現される方法であって、

少なくとも１つのデータファイルと署名ファイルとを受け取る工程を備え、該データファイルと該署名ファイルは別個であり、該データファイルは識別子を含み、該署名ファイルは該データファイルのための該識別子と電子署名とを含み、

該署名ファイルの真正性を判断するためにコンピュータシステムを使用して署名ファイルの処理する工程を備える、方法。

【請求項２】　請求項１に記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

該コンピュータシステムを使用して、該データファイル内の該識別子と該署名ファイル内の該識別子を比較し該データファイルの真正性を判定する工程を更に含み、署名ファイルを処理する該工程は、該署名ファイルの真正性を判定するためにコンピュータシステムを使用して該電子署名を処理する工程、を更に含む方法。

【請求項３】　請求項２に記載の方法であって、該データファイル内の該識別子と該署名ファイル内の該識別子が一致するとき、該データファイルを署名されたものとしてマーキングする工程を、更に含む方法。

【請求項４】　請求項２及び請求項３のいずれか１つに記載の方法であって、

該データファイルの該識別子と該署名ファイルの該識別子とが一致しないとき、該データファイルを無視すること、該データファイルのロードを中止すること、及びユーザに警告すること、の群から選択される少なくとも１つを、該データファイルの該識別子と該署名ファイルの該識別子とが一致しないとき、更に含む方法。

【請求項５】　データの真正性を検証するための、コンピュータで実現される請求項２～請求項４のいずれか１つに記載された方法であって、

該コンピュータシステムを使用して該データファイル内の該識別子を該署名ファイルの該識別子と比較する該工程が、第２のデータファイルに対して繰り返される、方法。

【請求項６】　先行する請求項のいずれか１つに記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

該電子署名を処理する該工程は、署名アルゴリズムを用いて該電子署名を検証する工程を更に含み、該署名アルゴリズムはキー付きのアルゴリズムであり、該署名アルゴリズムはＤＳＡアルゴリズムとメッセージ・ダイジェスト及びＲＳＡアルゴリズムの組み合わせとから成る群から選択される、方法。

【請求項７】　先行する請求項のいずれか１つに記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

2

該識別子は、１方向ハッシュ関数アルゴリズム及びサイクリック冗長チェックサムアルゴリズムの一方を使用して発生される、方法。

【請求項８】　先行する請求項のおずれか１つに記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

該データファイル内の該識別子を該署名ファイル内の該識別子と比較する工程は、１方向ハッシュアルゴリズムを用いて１つ以上の該識別子を発生する工程を、更に含む方法。

【請求項９】　先行する請求項のいずれか１つに記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

該データファイル内の該識別子を該署名ファイル内の該識別子と比較する該工程は、サイクリック冗長チェックサムアルゴリズムを用いて１つ以上の該識別子をチェックする工程を、更に含む方法。

【請求項１０】　先行する請求項のいずれか１つに記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

該データファイルと該署名ファイルとを受け取る該工程は、ネットワーク接続されたコンピュータ間で該データファイルと該署名ファイルを転送する工程を、更に含む方法。

【請求項１１】　先行する請求項のいずれか１つに記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

該データファイル内の該識別子は、証明の認証、サイト証明、ソフトウェア配布者の識別子、及びサイト名のうちの少なくとも１つを含み、

当該方法は、前記証明の認証、前記サイト証明、前記ソフトウェア配布者の識別子、前記サイト名の少なくとも１つに対してセキュリティレベルの設定を含む、方法。

【請求項１２】　請求項１１に記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

該データファイルを該コンピュータシステムにダウンロードする工程を含み、該データファイルがアプレットを備えるとき、そして該電子署名が検証されるとき、当該方法は、検証されたものとして該アプレットにブランドを付ける（brand）工程、及び該アプレットを動作させる工程を含む、方法。

【請求項１３】　請求項１２に記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

該データファイルがアプレットを備えるとき、そして該署名が検証されないとき、当該方法は未署名のデータファイルが該コンピュータ上での実行に対して受け入れ可能か否かを決定する工程、未署名のデータファイルが前

記コンピュータ上での実行に対して受け入れ可能でない場合に該アプレットを停止する工程、を含む方法。

【請求項１４】　請求項１３に記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

該未署名のデータファイルが前記コンピュータ上での実行に対して受け入れ可能と決定されるとき、該アプレットにブランドを付ける工程、を含む方法。

【請求項１５】　請求項１４に記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

アプレットを動作させる工程と、該アプレットがセキュリティチェックを駆動するアクションを実行するか否かを決定する工程とを含み、該セキュリティチェックは、該ブランドをセキュリティレベルと比較する工程、該セキュリティチェックが満たされるとき該アクションを許容しかつセキュリティチェックが満足されない場合該アクションを許容しない工程、を含む方法。

【請求項１６】　先行する請求項のいずれか１つに記載された、データの真正性を検証するための、コンピュータで実現される方法であって、

該コンピュータシステムを使用して遠隔のサイトとデータ通信の接続を確立する工程と、該サイトが安全な接続を要求しているか否かを決定する工程と、安全な接続が要求されているという決定に応じて、該サイトに対するサイト証明が妥当であるか否かを決定する工程と、を含む方法。

【請求項１７】　少なくとも１つのデータファイルと署名ファイルとの真正性を検証するための装置であって、該データファイルは識別子を含み、該署名ファイルは該データファイルのための該識別子と電子署名とを含み、電子署名を処理し該署名ファイルの真正性を決定するためのプロセッサを備え、

コンピュータシステムを使用して該データファイル内の該識別子を該署名ファイル内の該識別子と比較し該データファイルの真正性を判断するためコンパレータを備え、該プロセッサは該コンピュータシステムを使用して該電子署名を処理し該署名ファイルの真正性を決定するように更に配置されている、装置。

【請求項１８】　請求項１７に記載された、データの真正性を検証するための装置であって、該コンピュータシステムを使用して該データファイル内の該識別子を該署名ファイル内の該識別子と比較するための該コンパレータは、該データファイルの該識別子と該署名ファイルの該識別子とが一致するとき、署名されたものとして該データをマークするためのマーカを含む、装置。

【請求項１９】　データの真正性を検証する際の使用のためにコンピュータで利用できる媒体上に具体化されたコンピュータで可読なプログラムコードを有するコンピュータで利用できる媒体、を含んだコンピュータプログラム生産物であって、該プログラムは、コンピュータシステムで以下の

ａ）少なくとも１つのデータファイルと署名ファイルとを受け取ることであって、該データファイルは識別子を含み、該署名ファイルは該データファイルのための該識別子と電子署名を含み、

ｂ）コンピュータシステムを使用して該署名ファイルを処理し該署名ファイルの真正性を判断すること、をもたらすための、コンピュータで可読なプログラムコードを含む、生産物。

【請求項２０】　請求項１９に記載されたコンピュータプログラム生産物であって、

該コンピュータシステムを使用して該データファイル内の該識別子を該署名ファイル内の該識別子と比較し該データファイルの真正性を決定するための生産物であって、該署名ファイルを処理することは該コンピュータシステムを使用して該電子署名を処理し該署名ファイルの真正性を決定することを含む、コンピュータで可読なプログラムコード更に含む生産物。

【請求項２１】　請求項２０で述べられたコンピュータプログラム生産物であって、

該コンピュータシステムを使用して該データファイル内の該識別子を該署名ファイル内の該識別子と比較することは、該データファイル内の該識別子と該署名ファイル内の該識別子とが一致するとき、該データファイルを署名されたものとしてマークすることを含む、ためのコンピュータで可読なプログラムコードを更に含む生産物。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、一般に計算資源の間のデータの共有に関するものである。より詳細には、本発明は、コンピュータシステム上で処理されているデータの真正性（authenticity）を保証し（secure）また証明するための方法、装置、及び生産物に関するものである。

【０００２】

【従来の技術】インターネットといったネットワーク化されたコンピュータ環境の人気が高くなるにつれて、共有されている情報を安全な（secure）方法でネットワーク化されたコンピュータ間において伝送する方法を用意する要求が高まってきた。例えば、インターネットのユーザがデータの形式で情報を他のユーザに送るとき、受け手のユーザにとって、受け取ったデータが何らかの方法で変質されていなかったこと、さもなければ改変されていなかったこと、を検証することが有益なことがある。さらに、受け手のユーザは、受け取ったデータが詐称者からではなく送り手ユーザから実際に送られたことを検証することが有益であることを見いだすこともある。

【０００３】その結果、コンピュータネットワークや他

のデータリンクにわたって転送されたデータのセキュリティを増す方法やアルゴリズムが、一定の成果とともに、開発されまた展開されてきた。もっと安全な方法には、データを送るに先立ってその全部または一部を暗号化し、また同様にして、送られたデータを使うに先立ってそれを復号化することを含む傾向がある。そのような暗号化や復号化の技術には、例えばコンピュータシステムを用い「署名アルゴリズム」を走らせることによって、暗号化データをデータファイルに付加し、そのデータファイル内のデータを暗号化したり、さもなければ変形したりすることを含む。

【０００４】現在、今日使用されているいくつかの署名アルゴリズムがある。１つの著名な署名アルゴリズムは、実際には、メッセージ・ダイジェスト（Message Digest）アルゴリズムとＲＳＡ暗号アルゴリズムを組み合わせ（例えば、ＭＤ５とＲＳＡの組み合わせ、ＭＤ２とＲＳＡの組み合わせ等）である。ＲＳＡ付きのメッセージ・ダイジェストは、カルフォルニアのレッドウッド・シティ（Redwood City）にあるメッセージ・データ・セキュリティ（Message Data Security）社から入手可能である。他の有名な署名アルゴリズムは、ＤＳＡ暗号化アルゴリズムである。ＤＳＡ暗号化アルゴリズムは、米国政府から入手可能であり、署名アルゴリズムとして私的な団体によって限られた目的のために用いることができる。これらの署名アルゴリズムを、限定された詳しさで以下に説明する。

【０００５】ＲＳＡアルゴリズム付きのメッセージ・ダイジェストは、データファイルに付加できる「電子署名（デジタル署名、digital signature）」を生成する性能（capability）を含む。電子署名は、基本的には、ユーザが受け取ったデータファイルの出所（source）が真正であることを証明できるメカニズムである。電子署名は、典型的には、関連のデータファイルに伴って、他のユーザに対して発生され、また供給されることができるデータの特別な続き（sequence）である。ほとんどの署名アルゴリズムの裏にある基本的なコンセプトは、すべてのユーザ（例えば、個人や会社や政府など）が「私的キー（private key）」や「公開キー（public key）」の両方を含む「キーの対」を有していることである。キー（鍵）は、例えば、数の続きである。私的キーは、単一のユーザに割り当てられまたそのユーザによって秘密に保たれるも独自のキーである。私的キーは、割り当てられたユーザによって使用され署名アルゴリズムを用いデータファイルのための電子署名を創り出す。それに対して、公開キーは、典型的にはすべての他のユーザに対して利用できるようにされる。公開キーは、受け取ったデータファイルに関する電子署名が真正である（例えば、その電子署名がその私的キーで創り出された）ことを検証するために、これらの他のユーザによって用いられることができる。検証プロセスは、同じ署名アルゴリズムで達成される。原理的には、このような検証プロセスは、受け取ったデータの出所の真正性において比較的高い水準の信頼度を提供できる。

【０００６】電子署名発生アルゴリズムに加えて、データファイルが何らかの方法で変質されていなかったことの真正性を証明するために用いられるアルゴリズムも存在する。これらのアルゴリズムは、典型的には「一方向ハッシュ関数（one-way hashfunction）」として知られている。そのようなアルゴリズムの一例は、上にて紹介されたメッセージ・ダイジェストである。一方向ハッシュ関数は、通常はキーを必要としない。一方向ハッシュ関数は、典型的には、データファイルに挿入される付加的なデータを含んである。そのため、データファイルを受け取ったとき、そのハッシュ関数は、そのハッシュ関数が発生されて以来、そのデータファイル内のどのデータも改変されていないことを検証するために使用できる。しかしながら、ハッシュ関数は、誰がそれを送ったいった、関連したファイルについての何事もユーザは必ずしも推論できないという点で典型的には限定されている。多くの署名アルゴリズムが一方向ハッシュ関数を内部の構成部品として使っている点を言及しておく。

【０００７】インターネットといった比較的にオープンでセキュリティに欠けるネットワークに対しては、受け取ったデータファイルを使うに先立ってそれらが真正であること証明することはユーザにとって有益である。そのようなデータファイルには、制限されるものではないが、コンピュータプログラム、グラフィックス、テキスト、写真、音声、ビデオ、又はコンピュータシステム内での使用に適する他の情報などがある。データファイルのタイプがどんなものであっても、真正性の証明は、上述したような署名アルゴリズム又は暗号化アルゴリズムの類似タイプで達成されることができる。例示によれば、もしデータファイルがソフトウェアプログラムならば、そのプログラムがユーザのコンピュータをウィルスに感染させるトロイの木馬（Trojan Horse）を含んでいると困るので、ユーザのコンピュータシステムをソフトウェアプログラムにさらすに先立ってそのプログラムが信頼できる権限者によって送られたことが真正であると証明することをユーザが望むかもしれない。そのような場合においては、送り手のユーザは上述したように、データの真正性を証明できる。

【０００８】別な例は、受け手のユーザが、自分のコンピュータの画面に表示するに先立ってテキストファイル及び画像データファイル、又はテキストファイル若しくは画像データファイルが真正であると証明することを望む場合である。これは、望ましくない内容を有するテキスト及び画像の表示を制御するために有効である。例えば、親たちは成人ものの主題や資料に関係する絵やテキストに対する子供が行うあらゆるアクセスを制限したいかもしれない。これは、そのデータファイル（例えば、

テキストファイル又は画像ファイル）が信頼できる出所から来たことを検証することによって達成されることできる。同様に、テキストファイルや画像ファイルの供給者は、商標や他の知的所有権の使用を制御するように認可又は真正性の「スタンプ」を与えたいかもしれない。

【０００９】

【発明が解決しようとする課題】残念ながら、暗号化することと復号化すること、署名することと検証すること、及び／又はハッシュ関数を生成すること、のプロセスは、送り手及び受け手のユーザのコンピュータ資源に追加の負荷をかける（place）。この負荷は、いくつかのデータファイルを送ったりまた受けたりするユーザに対して度合いが増す。例えば、ワールドワイドウェブ（World-Wide Web）として知られるインターネットの側面が成長すると、ユーザ間における多数の（multiple）データファイルの転送において激増することに至る。このような多数のデータファイルは、Ｊａｖａ™アプレット（Java applet）といったオブジェクト指向のソフトウェアプロセスを構成するコンポーネント又はオブジェクトをしばしば含む。そのような多数のデータファイルの転送の際にユーザのコンピュータ資源上にかけ得る潜在的な負荷を例示するためには、そのファイルの各々のために電子署名を検証することに関連する結果として生じる処理時間を計算することのみを必要とする。例えば、仮にＪａｖａ™アプレットが電子的に署名された（データファイルも含む）２００個ものＪａｖａ™クラスファイルを含むとすると、通常のデスクトップコンピュータ上で平均検証時間が１秒であると仮定して、ユーザはデータファイルを受けた後にアプレットを使うために約２００秒もの間待たなくてはならないでしょう。そのような時間遅れは、コンピュータネットワーク環境の効率を著しく縮小させる可能性がある。このことは、リアル（またはリアルに近い）タイムでのストリーム方式の音声又はビデオのデータファイルといった、時間が決められた処理に関係するデータファイルに関してはまさに真実となる。

【００１０】従って、望まれていることは、データファイルの真正性を保証しまた検証するための、特にコンピュータネットワークにわたって転送されると意図されるデータファイルについての、より効率的な方法、装置および生産物である。

【００１１】

【課題を解決するための手段】本発明は、コンピュータネットワークをわたって転送されると意図されるデータファイルのような、データファイルの真正性を保証しまた検証するための、より効率的な方法、装置および生産物を提供する。本発明の１つの側面に従って、データの真正性を検証するための方法は識別子と署名ファイルとを含む少なくとも１つのデータファイルを提供することを伴い、この署名ファイルは電子署名だけでなく、データファイルのための識別子を含む。そして、電子署名はコンピュータシステムを使用して検証され、そのデータファイル内の識別子はコンピュータシステムを使って署名ファイル内の識別子と比較される。

【００１２】１つの実施例として、そのデータファイルのための識別子は、少なくとも１つの証明の認証、サイト証明、ソフトウェア配布者の識別子、またはサイト名を含み、またデータの真正性を検証することは少なくとも１つの証明の認証、前記サイト証明、前記ソフトウェア配布者の識別子、前記サイト名のためのセキュリティレベルを設定することを伴う。そのような実施例においては、データファイルがコンピュータシステムにダウンロードされ、そしてそのデータファイルがアプレットでありまたその電子署名が検証されると、データの真正性の検証することは、そのアプレットへブランドを付けることとそれに応じてアプレットを動作させることを伴う。

【００１３】本発明の別の側面では、識別子を含んだ少なくとも１つのデータファイルと、電子署名に加えてデータファイルのための識別子を含む署名ファイルとの真正性を検証するための装置は、電子署名を検証するベリファイヤ（verifier）と、データファイル内の識別子と署名ファイル内の識別子を比較するためのコンパレータ（comparator）とを含む。別の実施例においては、そのコンパレータは一方向ハッシュ関数アルゴリズムを含む。

【００１４】本発明のさらに別の側面においては、識別子を含み、且つデータファイルのためのその識別子と電子署名とを有する署名ファイルに関連づけられる、データファイルの真正性を検証するために用意されるコンピュータシステムは、プロセッサと、プロセッサに結合されるメモリと、電子署名を検証しまたデータファイル内の識別子を署名ファイル内の識別子を比較するために用意されたベリファイヤとを含む。ある実施例においては、そのデータファイルのための識別子は証明の認証、サイト証明、ソフトウェア配布者の識別子、およびサイト名のうちの少なくとも１つを含む。そのような実施例においては、ヴェリファイヤは証明認証、サイト証明、ソフトウェア開発者の識別子、およびサイト名のうちの少なくとも１つのためのセキュリティレベルを設定するように調整されている。別の実施例では、データファイルはアプレットであり、ベリファイアはアプレットにブランドを付け、且つそのアップレットを動作させるために用意されている。

【００１５】本発明とそれについてのさらなる利点を、添付図面と共に以下の記述を参照することによって最もよく理解できる。

【００１６】

【発明の実施の形態】本発明のいくつかの実施例は、任意の数のデータファイルに対して単一の電子署名のみの

実施とその検証を要求することによって、出所ユーザのコンピュータシステムと受け手ユーザのコンピュータシステムとの両方にかけられた計算要求を減少させる新規の方法、装置および生産物（product）を供給する。本発明の実施例に従うと、データファイルを個々に署名する必要がない。その代わりに、１つの別個の（separate）署名ファイルが電子的に署名され後に検証されるときに署名ファイルが対応するデータファイルがこれらのデータファイルの各々に対して署名アルゴリズムを実行することなく真正性を証明できるように、その別個の署名ファイルが創り出される。ある実施例においては、署名ファイルは、転送されるべき個々のデータファイルに関連付けられた、一方向ハッシュ関数といった、「識別子」のリストを含む。そのため、署名ファイルは、基本的には、データファイルの各々のための電子署名と暗号的に同等なものである。

【００１７】このため、本発明の１つの実施例に関して、ユーザは各データファイルのために独自の識別子を含む署名ファイルを創り出すことができる。この署名ファイルは、署名アルゴリズムを使って電子的に署名されることができる。そして、署名された署名ファイルとデータファイルとを受け手のユーザに送ることができ、そのユーザは適切な署名アルゴリズムを使ってその電子署名を検証できる。一旦、電子署名が検証されると、署名ファイル内にある識別子をデータファイル内の識別子に比較できる。与えられたデータファイル内の識別子が署名ファイル内の対応する識別子と一致するならば、真正であるとしてそのデータファイルは検証され得る。すると、受け手ユーザは、真正性について確信を持って、その検証されたデータファイルを処理することを進めることができる。その結果、電子的に署名しまた後にデータファイルの各々に対して電子署名を検証する必要がもはやないので、計算による遅れを縮小できる。

【００１８】図１は、データ形式の情報を受け手ユーザのコンピュータシステム１４と交換するためにデータリンク１６にわたって結合された出所ユーザのコンピュータシステム１２を、ブロックダイアグラムによって表現されるものとして、ネットワーク化された計算環境１０を示す。出所ユーザのコンピュータシステム１２は、例えば、インターネットに関連づけられたウェブサーバといったサーバコンピュータの形式をとることができる。同様に、受け手ユーザのコンピュータシステム１４は、例えば、データリンク１６にわたってウェブサーバにネットワーク接続されたクライアントシステムの形式をとることができる。そのようなケースでは、データリンク１６は、したがって、インターネットや他の接続されたネットワークの一部か、または、全体を表わすことができる。データリンク１６は、また、１つ以上のローカルエリアネットワーク（ＬＡＮｓ）、広域エリアネットワーク（ＷＡＮｓ）、「イントラネット」若しくは「エクストラネット」、又は他の同様の電子通信若しくはデータのネットワークを表わし得る。

【００１９】図２は、図１に従う、送り手ユーザまたは受け手ユーザのいずれかが使用できる典型的なコンピュータシステム２０を示す。代わりに、コンピュータシステム２０は、コンピュータに使える生産物を介してデータを受け取ることの能力があるスタンドアロンのコンピュータであっても良い。コンピュータシステム２０は、１つ以上のプロセッサ２２、主記憶２４、２次記憶２６、１つ以上の入力／出力（Ｉ／Ｏ）デバイス２８、１つ以上のネットワーク通信デバイス３０、１つ以上のバス３２を含む。

【００２０】プロセッサ２２は、コンピュータ命令を実行する能力を提供する。プロセッサ２２は、例えば、市場で入手できる多くのデスクトップコンピュータ、ラップトップコンピュータ、ワークステーション、メインフレームコンピュータ内で見られるようなマイクロプロセッサ、中央処理装置（ＣＰＵ）、マイクロコントローラであり得る。また、プロセッサ２２は、特定目的コンピュータ若しくは大型フレームコンピュータ、通信交換ノード、又は他のネットワーク化された計算デバイス内で典型的に使われるプロセッサといった従来のプロセッサ又はカスタマイズ若しくはセミカスタマイズされたプロセッサの形式をとることもできる。プロセッサ２２は、バス３２へのデータを出力するため、またバス３２からのデータを入力するために結合されている。

【００２１】バス３２は、２つ以上のノード間において、データを転送すること又はさもなければデータを移動することの能力を持っている。例えば、バス３２は、共有の汎用バスの形式をとることができ、又は特定ノード間において特定タイプのデータを転送するために専用にされることができる。バス３２は、データを転送できるノード間の経路を確立する際に使うためのインターフェース回路構成およびソフトウェアを含むことができる。プロセッサ２２といった、あるデバイスは、内部ノード間においてデータを転送するための１つ以上のバス３２を内部に含むこともできることが認識される。データは、処理済データ、アドレス、及び制御信号を含むことができる。

【００２２】主記憶２４は典型的には、データの格納および検索のために備えている。主記憶２４は、例えば、ランダムアクセスメモリ（ＲＡＭ）又は同様の回路であり得る。主記憶２４はバス３２を介して、プロセッサ２２といった他のデバイス又は回路によってアクセスされ得る。

【００２３】２次記憶２６は、典型的にはデータの付加的な格納及び検索のために備えている。２次記憶２６は、例えば、磁気ディスクドライブ、磁気テープドライブ、ＣＤＲＯＭといった光学的に読み取り可能なデバイス、ＰＣＭＣＩＡカードといった半導体メモリ、又は同

様のデバイスの形式をとることができる。２次記憶２６は、バス３２を介して、プロセッサ２２といった他のデバイス又は回路からアクセスされる得る。２次記憶２６は、例えばその上に具現化されたコンピュータで可読なコードを有するコンピュータで利用可能な媒体を含むコンピュータプログラム生産物からデータをアクセスし又は読むことができる。

【００２４】入出力デバイス２８は典型的には、ユーザに対するインタフェースを提供し、それを介してデータが共有される。入出力デバイス２８は、例えば、キーボード、タブレット及び指示ペン、音声認識器及び手書き文字認識器、又は別のコンピュータといったよく知られた他の入力装置の形式をとることができる。入出力デバイス２８は、例えば、ディスプレイモニタ、フラット・パネル・ディスプレイ、又はプリンタの形式をとることもできる。入出力デバイス２８は、バス３２を介して、プロセッサ２２といった他のデバイス又は回路によってアクセスされ得る。

【００２５】ネットワーク通信デバイス３０は典型的には、他のコンピュータシステムといった他の計算資源およびデバイスへのインタフェースを提供する。ネットワーク通信デバイス３０は典型的には、データ通信リンクおよびデータ通信リンクネットワークにわたってデータ通信の標準およびプロトコルを実現するインタフェースのハードウェア及びソフトウェアを含む。例えば、ネットワーク・コネクションを用いて、プロセッサ２２はネットワークにわたってデータ（すなわち情報）を送り及び受け取ることができる。上述したデバイス及び方法は、コンピュータのハードウェア及びソフトウェアの分野の当業者には精通していることである。

【００２６】図３（ａ）は、本発明の実施例に従う、アーカイブ（記録保管所）のデータ構造３００の実施例を示す。データ構造３００は、署名ファイル３０２といくつかのデータファイル３０４～３１４とを含む。ファイル３０４～３１４は、任意のデジタルビット列でもよく、例えば、Ｊａｖａ™クラスファイル、画像ファイル、音声ファイル、テキストファイル、および付加的な署名ファイルであってもよい。

【００２７】図３（ｂ）は、署名ファイル３０２の具体例である。ある実施例においては、署名ファイル３０２はヘッダファイルであることを留意すべきである。図示された実施例においては、署名ファイル３０２には、データファイル３０４～３１４の各々のための少なくとも１つの識別子３１６を含む。オプションとして、署名ファイル３０２は、データファイル３０４～３１４の各々のための付加的なデータ３１８も含むことができる。例えば、付加的なデータ３１８は、ファイルの名前、ファイル作成者、ファイルの日付、ファイルの版数、ファイルのレート付け（例えば、「親の指導が必要」といった映画のレート付け）、またはユーザが署名ファイル３０２内に含めたい真正性の証明される他の任意にデータについての情報を更に備える。

【００２８】署名ファイル３０２は、識別子ＩＤ３２０と電子署名３２２とを更に含む。識別子ＩＤ３２０は、署名ファイル３０２に列記された識別子を創り出すために用いられるアルゴリズムを決定するために必要な情報を提供する。電子署名３２２は、その署名ファイルのために創り出された電子署名を表わす。もちろん、電子署名３２２の構造は、それを創り出すために用いられる署名アルゴリズムに依存する。

【００２９】図４は、本発明の実施例に従い、１つ以上のデータファイルを発生するためのステップ４０２を含む方法４００を図示する。例えば、ステップ４０２は、テキストファイルを発生するためのテキストプログラム、音声やビデオのファイルを発生するための記録プログラム、画像もしくは映像のファイルを発生するためのグラフィックスプログラム、クラスファイルやプログラムファイルを発生するためのプログラミング言語、又はデータファイルを発生する能力がある他の任意の機構を使用することを含むことができる。

【００３０】ステップ４０２で１つ以上のデータファイルを発生したら、ステップ４０４はこれらのデータファイルの各々について識別子を発生する。例えば、ステップ４０４で生成された識別子は、一方向ハッシュ関数アルゴリズムによって発生されることができ、また代わりにサイクリック冗長性チェックサム（ＣＲＣ、cyclicredunduncy checksum）等の形式をとることもできる。しかしながら、一般的に一方向ハッシュ関数アルゴリズムはより大きなセキュリティに備える傾向にあり、なぜなら、そのような関数は簡単には又は効率的には破られることができず、またさもなければリバースエンジニアされることができないからである。例として、ＭＤ５及びＳＨＡといった一方向ハッシュ関数アルゴリズムは典型的には、暗号的に安全であると考えられる。そのようなアルゴリズムは、コンピュータ科学の分野の当業者に知られるであろう。

【００３１】次に、ステップ４０６は、ステップ４０４で発生されるような識別子を列記し、またはコンパイルする署名ファイルを創り出すことを含む。署名ファイルは、例えば、識別子を列記するテキストファイルであってもよい。オプションとして、署名ファイルは、例えば、各ファイルの名、各ファイルの著者名、ファイルの版数、ファイルの日付スタンプ、各データファイルに関係する他のデータを更に含んでもよい。ステップ４０６はさらに、データファイルからのそのようなデータを問い合わせ、たどり（トレースし、trace）、選択し、集め、又は与える（render）１以上のプログラムを含むことができる。ステップ４０６は、例えば、バッチモードでデータファイルを処理し適当な識別子及び付加的なデータを集めることによって、実行できる。当業者は、方

法４００におけるステップを促進するある方法で、署名ファイルに列記されるデータを特定的に、順番に並べ、グループ分けし、又は配置することが、（例えば、効率の面で）有益であることを認識するであろう。例えば、識別子に加えてファイル名及び作成者をグループ分けすることは有益であろう。

【００３２】一旦、署名ファイルが創り出されたら、ステップ４０８は署名アルゴリズムを使って署名ファイルに電子的に署名することを含む。適切な署名アルゴリズムの例示は、メッセージダイジェストアルゴリズムおよびＲＳＡ暗号化アルゴリズムを組み合わせたもの（例えば、ＲＳＡを持つＭＤ５、若しくはＲＳＡ持つＭＤ２等）、または（上記で説明した）ＤＳＡアルゴリズムを含む。ステップ４０８は、例えば、公開キー又は私的キーを手段として署名アルゴリズムを用いて署名ファイルのための電子署名を発生することを含む（例えば、上記のシュナイヤ（Schneier）を参照せよ）。

【００３３】ステップ４０８からの署名された署名ファイルは、次に、ステップ４１０において受け手ユーザに送られ、提供され、または利用できるようにされる。例えば、ステップ４１０は、署名された署名ファイルをデータバス、データリンク、インターネット、他のコンピュータ若しくはデータ通信のネットワーク若しくはリンクをにわたって転送されることを含むことができる。加えて、ステップ４１０は例えば、署名ファイルを磁気記憶媒体や光学的記憶媒体のようなコンピュータで読み込める媒体に格納すること、及びコンピュータで読み込み可能な媒体上の署名された署名ファイルをあるコンピュータから別のコンピュータに移動することを含むことができることが認識される。

【００３４】受け取り又はアクセスのときに、ステップ４１２における受け手ユーザは、ステップ４１０において送られ、又は利用できるようにされた署名された署名ファイルの真正性を検証する。ステップ４１２は例えば、キーを手段として署名アルゴリズムを用いて、署名された署名ファイル上の電子署名を検証することを含むことができる。

【００３５】ステップ４１４は、ステップ４１２で決定されたものとして電子署名の有効性（validity）が方法４００を終了するか又は継続するかのいずれかである決定を示す。方法４００を中断するとか又は回避する（preempt）というように表示がされる一方で、また署名済の署名ファイルのステップ４１２におけるその検証が失敗したことを、ある方法で記録し若しくは識別し、または発する（address）別のプロセス、例えば警告若しくは通知プロセス、またはログプロセス、を呼びだすことをステップ４１４は含むことができということが認識される。

【００３６】ステップ４１４での決定がそのファイルが正当である（すなわち、真正である）場合、プロセスは、署名ファイルからの少なくとも識別子を格納することを含むステップ４１６へ続く。本発明のある実施例においては、識別子がセキュア地帯（secure location）に格納される。セキュア地帯は、例えば受け手のコンピュータシステムのＲＡＭであってもよく、なぜならこのメモリはプロセスが終了するときに容易にクリアされるからである。代わりに、識別子はディスクドライブまたはテープドライブに格納されることもでき、そこでは識別子はある後のステージで検索できる。当業者は、いろいろなデータ記憶デバイス及び他のコンピュータシステム構成が様々な及び潜在的なセキュリティリスク（つまり、ある記憶デバイスは他よりもより安全である）を引き起こす（pose）ことを認識するであろう。また、ステップ４１４で記憶された署名ファイルの信用性を増しまた保証するために、暗号化やファイルアクセスの特権といった、さらなるセキュリティの方策（measure）を使うできることも認識できる。

【００３７】ステップ４１６で識別子がセキュア地帯に記憶されてしまうと、次に対象のデータファイル、またはステップ４０６において署名ファイルにその識別子が列記されたデータファイルがステップ４１８に表示されるようなループに合わせて処理されることができる。ステップ４１８は、例えば、署名ファイルに列記されている識別子の数に基づいて、ステップ４２０に入る回数を繰り返し制御するカウンタ機構を含むことができる。例えば、署名ファイル列記されたｎ個の識別子がある（つまり、ロードされるべきｎ個のデータファイルがある）とならば、繰り返しループは、ｉ＝１からｉ＝ｎまで数え上げし若しくは代わりにｉ＝ｎからｉ＝１に数え上げすることができ、又は、以下に提供されるように方法４００内の残りのステップに合わせて、データファイルの全てがロードされたとき若しくはそのロードすることが試みられたときを決定する。

【００３８】ステップ４２０は、ｉ番目のデータファイルをロードするすることを含む。ステップ４２０は、例えば、ダウンロードし、アップロードし、同時通信し（broadcast）、またはさもなければｉ番目のデータファイルをある場所から別の場所に移動するいずれかのためにステップ４１０内の任意の方法を使用できる。一旦、ｉ番目のデータファイルがロードされたら、ステップ４２２は（そのデータファイルのために）適切な識別子アルゴリズムを使ってｉ番目のデータファイルのための識別子を提供し、計算し、または発生することを含むことができる。

【００３９】次に、ステップ４２４は、ステップ４１６で格納された署名ファイル内に、ｉ番目のデータファイルに関して、列記された識別子と、ステップ４２２で提供された識別子を比較することを含む。識別子が一致すると、そのときｉ番目のデータファイルを真正であるものとして検証する。識別子が一致しないならば、そのと

きｉ番目のデータファイルは検証されなかったとみなす。

【００４０】ステップ４２６は、ステップ４２４で決定されるような識別子の正当性がステップ４１８の繰り返しループを中断するか継続するかの判断を表示する。ｉ番目のデータファイルのための識別子がステップ４２４で検証されたなら、ステップ４２６は、ｉ番目のデータファイルが真正であるとして検証されたことを、マーキングし（印を付け、marking）、または何らかの方法で記録し若しくは確立することを含むステップ４２８へ進むことによって、ステップ４１８の繰り返しループを継続する。ステップ４２８は、例えば、ｉ番目のデータファイルを送り手ユーザによって署名されたものとして、修正しまたはマーキングすることを含むことができる。

【００４１】一方、ｉ番目のデータファイルのための識別子が真正なものとしてステップ４２４で検証されることができないと、そのときステップ４２６はステップ４３０に進むことによってステップ４１８の繰り返しループを中断する。ステップ４３０は、ステップ４２８を避けまたステップ４１８に逆進するように、ある方法でステップ４１８の繰り返しループを中断することを含む。ステップ４３０は、例えばｉ番目のデータファイルを無視することを含むことができる。ステップ４３０に加えて、ｉ番目のデータファイルが真正でないことをともかく記録しまた識別するために、別のステップを方法４００に含むことができる。

【００４２】このように、上記のデータ構造およびステップを用いて、いくつかのデータファイルを送っているユーザは、各データファイルに対して個別の電子署名を発生する必要があるというよりもむしろ、送り手ユーザが１つの署名ファイルを創り出しまたそのファイルに電子的に署名するだけでよいので、関連する処理時間を縮小できるでしょう。同様に、上記のデータ構造やステップを用いて、いくつかのデータファイルを受け取っているユーザは、関連する電子署名の暗号を解くことによって真正であるとして各データファイルを検証しなければならないというよりはむしろ、その署名ファイルの真正であることを検証するのみ必要であるので、関連する処理時間を減少できるでしょう。そのような混成の（ハイブリッドな、hybrid）検証プロセスは、署名プロセスと検証プロセスを実質的に合理化できる。その結果、データファイルは、電子署名され、そして後に短い時間で真正性が証明されまた処理される。

【００４３】加えて、ステップ４３０は，ステップ４２４において真正性を検証するために、試みられているロードを中止するオプションのステップ４３２、及び／又はその失敗を警告しまた警報を出すオプションのステップ４３４、へ導くことができる。一旦、ステップ４３０とオプションのステップ４３２及び／又はステップ４３４は完了すると、方法４００はステップ４１８に戻り、その中の繰り返しループを完了する。ステップ４１８の繰り返しループが完了したら、方法４００を終了する。

【００４４】本発明の実施例においては、署名した各認証のために、証明（certificate）が創出され、つまり署名ファイル内に列記される独自の識別子が証明として具体化される。一般的には、証明とは典型的には送り手コンピュータシステムかまたは受け手のコンピュータシステムのいずれかであるサイトが自分自身を識別するために使用できるトークンである。いくつかのサイトは１つの証明に関連付けられることができる。代わりに、いくつかの証明が１つのサイトに関連付けられことができる。

【００４５】出所のユーザのコンピュータシステムや受け手ユーザのコンピュータシステムは、データファイルだけでなく、例えばカリフォルニア、マウンテンビューのサンマイクロシステム（Sun Microsystems of Mountain View, California）から入手できるＪａｖａ™プログラミング言語で書かれた「アブレット」の形式であるコンピュータソフトウェアを交換するように構成されることができる。ここで使われた「アブレット」は、出所のコンピュータ又は典型的にはサーバからクライアントのマシンに伝えられる（pass）ように構成され、また例えばブラウザのような、既にクライアントにインストールされているソフトウェアと連携して実行される、ソフトウェアプログラムである。記述されている実施例では、アブレットはクラスファイルから具現化された（instantiate）ものであり、クラスファイルは、図３（ａ）に関して述べられたように、アーカイブのデータ構造にまとめられグループ化され、出所のコンピュータ、つまりサーバ、からクライアントのマシンにダウンロードされる。典型的には、アブレットは、実行するようにブラウザソフトウエア自身が構成されていない様々な計算タスクを実行することによって、ブラウザソフトエアに追加の機能を提供する。このため、アブレットをダウンロードするユーザは、そうしなければブラウザソフトエアに利用可能でない追加の機能をブラウザソフトウエアに提供できる。そのような追加の能力は、例えばデータベースに対するカスタムインタフェースを含む。

【００４６】与えられたアブレット、例えばＪａｖａ™アブレットがアクセスできるオペレーションを制御するために、出所コンピュータおよびクライアントマシンのいずれか上で、ブラウザに関連したセキュリティマネージャを使用できる。換言すれば、セキュリティマネージャは、アブレットが実行することを許可するアクションを制御するために、またはアブレットに対する特権を拡張するために、使用されることができる。アブレットに許可されるアクションは大きく変化しうるが、一般的には、そのアクションは読むおよび書くアクションである。アブレットに関連する異なる証明およびサイトに対する許可を設定するための柔軟性をユーザに与えるため

に、セキュリティマネージャの内部において、異なるセキュリティレベルを実現できる。一般的に、ユーザは特別の証明若しくはサイト、または一群の証明およびサイトを選択でき、そしてそのユーザの選択に関してのセキュリティレベルを設定できる。

【００４７】セキュリティレベルを実現するためにセキュリティマネージャを使うことは、一般的には、安全でなく、安心できず、若しくは信頼できないアプレットアクションだけでなく、どのアプレットアクションが安全であり、安心でき、若しくは信頼あできるかを、識別することにつながる。安全なアクションは、一般的には、システムセキュリティを危険にすること、またはクライアント若しくはサーバ上に格納されている情報が衝突する（corrupt）こと、に関して重大な可能性があるとみなされない。例として、安全なアクションは読み出し専用アクション、または特定のディレクトリに対する読み出し専用アクションがあり得る。一方、安全でない（unsecure）アクションは、一般的にはシステムセキュリティを違反することに関して、またはたクライアント若しくはサーバに格納されている情報に損害を与えることに関してあらゆる可能性のあるアクションである。安全でないアクションは、制限するものでないが、書き込みアクション、削除アクション、名前替アクション、および機密文書へのアクセスを要求するような読み出しアクションを含むことができる。安全でないアクションは、保護（プロテクト、protect）される遠隔サイトへの接続の確立するための要求を更に含む。

【００４８】本発明の実施例として、Ｊａｖａ™アプレットを実行できるブラウザは、例としてはＨｏｔＪａｖａ™ブラウザ（カリフォルニア、マウンテンビューのサンマイクロエレクトロニクスから入手可能）といった、高セキュリティレベル、中セキュリティレベル、低セキュリティレベル、不信頼レベルを含むセキュリティレベルを持ったセキュリティマネージャを有している。高セキュリティレベルにすれば、あらゆる安心でないアクションを遮断する一方で、基本的にはアプレットは一組の安心できるアクションと共にまたは制約と共に動作することが可能になる。記述された実施例においては、高セキュリティレベルにすれば、安心でないアクション、例えば信頼できないと考えられるどのアクションに対するアクセスも拒否する一方で、アプレットが安全であるアクション、例えば信頼できると考えられるほとんどのアクションを実行することが可能になる。

【００４９】中セキュリティレベルは、潜在的に安心でないアクションに対する認証を認める能力（ability）をユーザに提供する一方で、アプレットが安心な制約と共に動作することを可能にするために使用され得る。中セキュリティレベルを用いると、安心できる、例えば許容可能な、アクションでないかもしれないアクションをユーザインタフェースを介してユーザに警告できる。記述された実施例においては、アクティビティ（活動、activity）を記述する対話ボックスが現われ、ユーザは、実行される潜在的に安心でないアクションに対する許可を拒否することまたは認めることのいずれかを促される。低セキュリティレベルにすると、アプレットが最小の制約と共に動作し、また記述された実施例では、潜在的に安心でないアクションをユーザに警告しない。不信頼セキュリティレベルは、安心でないと知られている証明及びサイトを識別するために用いられる。

【００５０】次に図５を参照して、本発明の実施例に従って、セキュリティマネージャにおけるセキュリティレベルを設定することと関連づけられるステップが記述される。ある実施例では、セキュリティレベルは、信頼及び検証の設定のレベルとしても知られているが、前述したように、高セキュリティレベル、中セキュリティレベル、低セキュリティレベル、及び不信頼セキュリティレベルである。セキュリティレベルは、適切なグラフィカルユーザインタフェース（GUI）の使用を通してユーザによって設定されることができるが、セキュリティレベルを設定するために任意の適切な方法を使用できることを認識すべきである。

【００５１】セキュリティマネージャ５００においてセキュリティレベルを設定するプロセスが始まり、そしてステップ５０２で証明の認証（authority）に関するセキュリティレベルが設定される。証明の認証は、個々の証明と証明の群との両方に適用されるべき異なるセキュリティのレベル、つまり優先度を可能にする。一般的には、証明の認証は、限定されるものではないが、どのように特定の証明が使われるかを識別する情報を含む。例として、証明の認証は、ある証明が他の証明を保証しまたは真正性を証明することを可能にするために設定されることができる。

【００５２】証明の認証のためのセキュリティレベルが設定されると、ステップ５０４でサイトの証明のためのセキュリティレベルが設定される。サイトの証明は、トランザクション処理（transaction）を成す安全なコネクション（connection）を開始するために、所与のサイトが使用できる証明である。サイトの証明のためのセキュリティレベルは、一般的に、セキュア・ソケット・レイヤ（secure socketlayer：ＳＳＬ）標準プロトコルと、安全なトランザクションが起こるべきコネクションの真正性を証明するために使用できるセキュリティの認証とを明記することを伴う。そのような安全な通信の技術が、悪漢のサイト若しくは潜在的に安全でないサイトを識別するため、またそれ故、そのようなサイトとの通信を避けることによって伝達できるより安全なチャネルを提供するために使用できる。

【００５３】ステップ５０６では、ソフトウェア配布者のためのセキュリティレベルが設定される。イントラネットの環境においては、イントラネットは安全な環境で

あると通常は考えられているので、ソフトウェアは通常は証明と共に配布されない。それ故、そのような安全な環境で配布されたソフトウェアは、一般に安全であると仮定される。しかしながら、例えばインターネット環境といった、ソフトウェアが証明と共に配布される環境に関しては、その証明に関連付けらたソフトウェアコードがブラウザの実行に対して信頼できるかどうかを決定するためにその証明を使用できる。

【００５４】サイト名のためのセキュリティレベルが、ステップ５０８で設定される。サイト名のためのセキュリティレベルの設定のプロセスは、サイトのためにセキュリティレベルが設定されるときそのセキュリティレベルはそのサイトに関連するすべてのソフトウェアに適用されるという点を除いて、ソフトウェア配布者の設定のプロセスと本質的に同じである。サイト名の許可を設定することは、一般にはシステムの資源にみだりに変更する（tamper）ほとんどないリスクと共に、証明のないソフトウェアをテストすることを可能にする。次に、証明タイプがステップ５１０で設定される。証明タイプを設定すると、どのように証明が使われるべきかを決定すること、またどのように証明が使われることを期待するかに基づいて証明のための認証を選択することを伴う。証明サイトがステップ５１０で設定された後に、セキュリティレベルの設定のプロセスが終了する。セキュリティレベルが設定される順番が、特定のセキュリティマネージャの要求に依存して大きく変わり得ることを認識すべきである。

【００５５】ある実施例においては、セキュリティレベルを設定するために、追加の「高度な（advanced）」設定を使用できる。１つの実施例では、高度の設定は、個人の証明認証、サイト証明、ソフトウェア配布者、又はサイト名のための特定の注文で特製にされた（カスタマイズされた、customized）セキュリティレベルをユーザが設定することを可能にするために、グラフィカルユーザインターフェース（ＧＵＩ）まはた類似したインターフェースを通してユーザが変更できる細分性（グラニュラリティ、granularity）の制御である。さらに、高度な設定は、一群の証明の認証、サイト証明、ソフトウェア配布者、またはサイト名のためのセキュリティレベルをユーザがカスタマイズすることを可能にするように構成できる。このため、高度なセッティングにすると、一般的に、セキュリティレベルを制御する際に、また全体的な証明の取り扱いの際に柔軟になる。例として、高度な設定の使用を介して、ユーザは特定のサイト証明を中セキュリティレベルに設定でき、一方ではサイト証明に関連するセキュリティ許可を読み出しアクセスのみの許容に制限することを明記できる。加えて、高度なセッティングを使用し、所与のセキュリティレベルにおける仕様を置き換え（override）でき、例えば高度なセッティングを使用し所与のセキュリティレベルでは通常は許されていないような許可を認める（grant）ことができる。

【００５６】記述された実施例においては、高度な設定は、上述したように高セキュリティレベル、中セキュリティレベル、低セキュリティレベル、および不信頼レベルに加えて、サイト証明、例えば選択されたセキュリティレベルに加えて実施されることができるサイト証明、について特定の許可をユーザが選択することを可能にするために付加的なオプションが利用可能であるように、提供される。これらの特別な許可は、これらに限られるわけではないが、ユーザに警告メッセージを与えずにアプレットがウィンドウを開けることを許したり、警告の対話を用いまたｈ用いずにアプレットが自動的に局所的なアプリケーションを始める（launch）ことを許したり、警告の対話を与えることなしにアプレットがすべてのプロパティにアクセスすることを許したり、また警告の対話を与えることなしにアプレットが実行を開始することを許したりするこを含むことができる。

【００５７】図６は、本発明の実施例に従った高度の設定を例示するブラウザインターフェースを図式的に表す。前に述べたように、高度な設定は、ブラウザインタフェースと共に提供されるセキュリティレベルに加えて、可能にしたり不可能にしたりできる特定の許可を選択するために使用される。ブラウザインタフェース５６０は任意の適当なブラウザインタフェース５６０であってもよいが、記述された実施例においては、ブラウザインタフェース５６０はＨｏｔＪａｖａ™ブラウザの基本的な表現となっている。示されるように、ブラウザインターフェース５６０には高度な設定のディスプレイ・ウィンドウ５６４がある。ディスプレイ・ウィンドウ５６４の第１の領域５６８は、サイト及び証明５７０とサイト及び証明の群とを列記して、それらのセキュリティ許可をカスタマイズする。 "Applet Permissions"（アプレットの許可）５７２、 "File Access"（ファイルアクセス）５７４、"Network Access"（ネットワークアクセス)５７６は、カスタマイズできる許可の中にある。 記述された実施例では、選択５８０は、"File Access"（ファイルアクセス）５７４、つまりアプレットがアクセスすることを許容するファイル、に関する設定を示す。

【００５８】第２のサブ領域５８２は、選択されたときに、選択された設定、つまり"Applet Persmissions"（アプレットの許可）５７２のコマンドを使って決定される設定、を持つアプレットが読み取ることを許すファイルとディレクトリ５８４を表示する。同様に、第３のサブ領域５８８は、適切な許可を持つアプレットが書きこむことが許すファイルやディレクトリ５９０を表示している。「"Warn beforegranting access to other files"（他のファイルへのアクセスを許す前に警告せよ）」オプション５９４、又は「"Warn when applet tr

ies to delete a files"（アプレットがファイルを削除しようとするときに警告せよ）」オプション５９６の例のような付加的な選択可能なオプションにすると、ユーザがセキュリティオプションを更にカスタマイズすることを可能にする。

【００５９】図７は、本発明の１つの実施例に従う、検証の設定を用いるアプレットを実行する一つのプロセスに関連付けられたステップを図示するフローチャートである。検証の設定を実施するプロセスが開始し、ステップ６０２において、アプレットは、それが実行されるべきローカルなマシンへダウンロードされる。記述された実施例においては、アプレットをダウンロードすることは、クラスファイル、それからそのアプレットが具体例をあげて示される（instantiate）ファイル、を含むアーカイブファイル、つまりアーカイブデータ構造、の少なくとも一部をダウンロードすることを伴う。そのアプレットがダウンロードされた後、署名されたアーカイブストリームがステップ６０４で受け取られる。記述された実施例においては、アーカイブストリームはＪａｖａ™アーカイブファイルに関連した電子署名を含む。

【００６０】ステップ６０６では、その署名されたアーカイブストリーム内の署名が妥当（valid）であるか、つまり、その署名が既知の且つ受け入れ可能なものであるか、に関して決定がなされる。署名されたアーカイブストリームは前述した署名された署名ファイルの一実施例であることを認識すべきである。アーカイブストリーム内の署名が妥当であるかどうかの決定は、既知の認証を見つけるまで、署名ファイル内に認証、例えば証明、の一続き（chain）を体系的にチェックすることを伴う。そして、既知の認証が妥当性に関してチェックされる。例として、証明”Ａ”は、妥当な証明であると知られている証明”Ｂ”によって保証できる（vouch）。すると、証明”Ｂ”が妥当であると知られているので、証明”Ａ”が妥当であると決めてかかることができる。

【００６１】決定がその署名が妥当であるということならば、プロセスフローは、アプレットに「ブランドを付ける（brand）」ステップ６０８に進む。アプレットにブランドを付けしまたはマーキングすることは、一般的にはそのアプレットに署名者を結び付ける（attach）か、または、そのアプレットの妥当性を識別するために使用できる識別子をアプレットに結び付けることを指す。一旦、アプレットに適切にブランド付けされたら、そのアプレットはステップ６１０で実行される、つまり動作する。そのアプレットが動作している中、そのアプレット内の様々なアクションが呼び出される。

【００６２】ステップ６１２で、そのアプレットが動作を終了したかどうかに関する決定がなされる。言い換えると、そのアプレットに関連する各アクションが実行されたか、または、以下に述べるように実行することを許可されないか、のいずれかが決定される。アプレットの実行が完了したことが決定されたら、アプレットを実行するプロセスは終了する。アプレットの実行が終了していないこととが決定されると、ステップ６１４で、そのアプレットのアクションがセキュリティチェックを駆動する（trigger）かどうかが決定される。すなわち、ステップ６１４では、特定のアプレットのアクションがユーザのシステムのセキュリティに潜在的に害となる決定されたアクションの範囲に納まるかどうかの決定がなされる。そのようなアクションは、コンピュータ技術分野の当業者、特に、コンピュータセキュリティ技術の当業者とって、よく知られている。例として、ユーザのシステムセキュリティに潜在的に害であるアクションは、制限されるものではないが、制約されていない書き込みアクセス、システム資源の変更、及び他のシステムへのオーブンな伝達を含む。

【００６３】アプレットのアクションがセキュリティチェックを駆動するなら、ステップ６１６において、ステップ６０８でそのアプレット上に置かれた（place）ブランドは前にユーザによって与えられたセキュリティ設定、すなわち許可レベル、と比較される。ある実施例においては、セキュリティ設定をアプレット上のブランドと比較することは、ユーザインタフェースを介してユーザに相談することを伴う。そのような実施例においては、ユーザはセキュリティ設定の迂回路（バイパス）を正当と認めることができ、すなわち、ユーザはセキュリティ設定を置き換えし（override）特定のアクションを許し又は否定することのいずれかができる。ステップ６１６から、プロセスの制御は、アプレットアクションのセキュリティが満足されているかどうかの決定であるステップ６１８に進む。セキュリティが満足されていると、そのアプレット上に置かれたブランドがセキュリティ設定と好ましくは照合するという事実、または、ユーザがそのアプレットのアクションを正当と認めたという事実、という理由（virtue）のいずれかに基づき、ステップ６２０でそのアプレットのアクションが許される。すると、プロセス制御はステップ６１０に戻り、そしてアプレットの実行が継続する。一方、ステップ６１８でセキュリティが満足されていないと、アプレットのアクションはステップ６２２で許可されない。アプレットのアクションが不許可にされた後、プロセス制御はアプレットの実行が継続されるステップ６１０に戻る。

【００６４】アプレットのアクションがステップ６１４でセキュリティチェックを駆動しないと、プロセスフローはアプレットが動作することを継続するステップ６１０に戻る。プロセスフローは、アプレットが実行することを終了するか、または、現在のアプレットのアクションがセキュリティチェックを駆動するという決定がステップ６１４でなされ、その場合に前述のようにステップ６１８に進むか、のいずれかまで、ステップ６１０と６１４の間のループを継続する。

【００６５】ステップ６０６における署名の妥当性のチェックに戻って、署名が妥当でないという決定がされると、そのアーカイブは署名されていないとみなされて、そしてプロセスのフローは、未署名のストリームを受け入れるかどうかの決定がなされるステップ６２４へ進む。記述された実施例では、未署名のストリームが受け入れられるべきかどうかの決定が、ユーザインタフェースの使用を介してユーザによってなされる。例として、署名が妥当でないけれどユーザがそのアプレットを動作させるという決定をすることができることを表示する警告の対話を用いて、ユーザを促すことができる。未署名のストリームが受け入れられるべきだという決定がなされると、プロセスフローはステップ６０８に移り、そこではアプレットは適切であるとしてブランドが付けられ、例えばそのアプレットに関連づけられるストリームが未署名であることを示すためにブランド付けされる。未署名のストリームが受け入れられるべきでないという決定がステップ６２４でなされると、そのアプレットは停止させられ、つまりステップ６２６において動作することを禁止させられて、そしてアプレットの実行プロセスは終了する。

【００６６】次に、図８を参照して、コンピュータネットワークにわたる接続を達成することに関連するステップが、本発明の実施例に従って説明される。接続を成すプロセス７００は、望まれている接続が定義されるステップ７０２で始まる。記述された実施例では、望まれている接続を定義することは、接続が望まれているサイトのための汎用参照言語（ＵＲＬ）アドレスを特定することを伴う。接続が定義された後、ステップ７０４において、接続が望まれているサイトへ通信が確立される。

【００６７】ステップ７０４から、プロセスの制御は、サイトが安全な接続を要求しているかに関する判断が成されるステップ７０６に進む。ある実施例では、安全な接続は当業者によって理解されているように、セキュリティソケット層（ＳＳＬ）越しの（over）接続である。安全な接続が要求されていないことが決定されると、そのサイトへの接続がステップ７０８でなされ、そして接続を達成するプロセスは完了する。

【００６８】サイトが安全な接続を要求していることが決定されると、ステップ７１０において、そのサイトに関連するサイト証明が妥当なものであるかどうかが決定される。サイト証明が信頼されていないサイト、つまり安心でないことが既知であるサイト、に関する妥当な証明であることもあるから、妥当なサイト証明は必ずしも信頼されたサイト証明ではないことを認識すべきである。サイト証明が妥当であれば、プロセスフローはステップ７１２に進み、それはそのサイト証明が信頼できるものかどうかの決である。そのサイト証明が信頼できるという決定がなされると、プロセスフローは、そのサイトへの接続がなされるステップ７０８に進む。また、サイト証明が信頼できないという決定がなされると、ステップ７１４で、そのサイトと（ステップ７０４で）達成されていた通信は終了する。同様に、サイト証明が妥当でないという判断がステップ７１０でなされると、そのサイトとの通信はステップ７１４で終了する。

【００６９】上述したような、本発明の実施例は、コンピュータシステムに格納されたデータを伴う様々なプロセスステップを採用している。これらのステップは物理的な量に物理的な操作（manipulation）を要求するステップである。通常、必ずしもそうではないが、これらの量は、格納されたり、転送されたり、組み合わされたり、比較されたり、その他の操作されることが可能な電気的信号または磁気的信号の形態をとっている。ときには、主に一般の用法の理由に関しては、ビット、値、要素、変数、文字、データ構造などとして、これらの信号を参照することが便利である。しかしながら、このような用語また類似の用語の全ては、適当な物理量と関連付けられるべきものであり、またこれらの物理的な量に適用される単なる便利なラベルに過ぎないことを覚えておくべきである。

【００７０】さらに、実行される操作はしばしば、発生すること、算定する（caluculate）こと、計算する（compute）こと、マーキング、無視すること、中止すること、警報すること、検証すること、署名すること、送すること、受け取ること、創り出すこと、繰り返すこと、識別すること、動作させること、比較することといった用語で参照される。本発明の実施例の一部分を構成するオペレーションのどれかにおいては、これらのオペレーションはマシンのオペレーションである。本発明の実施例のオペレーションを実行する有効なマシンは、汎用のデジタルコンピュータか、または他の類似のデバイスを含む。どの場合でも、コンピュータを操作するオペレーションの方法と計算自体の方法との区別を心に抱くべきである。本発明の１つの実施例は、電気的または他の物理的な信号の処理においてコンピュータを操作し他の望まれた物理的な信号を発生するための方法のステップに関連する。

【００７１】本発明の実施例はまた、これらのオペレーションを実行するための装置（apparatus）に関連する。この装置は、要求された目的のために特別に構築されたものであってもよいし、コンピュータ内に格納されたコンピュータプログラムによって選択的に起動され（activate）または再構成される汎用のコンピュータであってもよい。ここで提示されたプロセスは、本来、特定のコンピュータまたは他の装置に関連していない。特に、ここであげた技術に従って書かれたコンピュータプログラムと共にさまざまな汎用のコンピュータを使用できるし、また、もっと特殊な装置を構築し要求された方法のステップを実行することはもっと便利である。さまざまなこれらのマシンのための要求構造は、上述した記

述から明らかである。

【００７２】加えて、本発明の実施例は、さまざまなコンピュータで実施される（implemented）オペレーションを実行するためのプログラム命令を含む、コンピュータで可読な媒体にも更に関連している。この媒体およびプログラム命令は、本発明の実施例の目的のために特に設計されまた構築されたものでもよく、または、それらはコンピュータソフトウェア技術分野の当業者によく知られまた利用可能な種類のものであってもよい。コンピュータで可読な媒体の例は、制限されるものではないが、ハードディスク、フロッピーディスク、及び磁気テープといった磁気的メディア、ＣＤ－ＲＯＭディスクといった光学的メディア、光ディスク（floptical disk）といった磁気光学的メディア、並びに読み出し専用メモリデバイス（ＲＯＭ）及びランダムアクセスメモリ（ＲＡＭ）といったプログラム命令を格納しまた実行するために特に配置されたハードウェアデバイスを含む。プログラム命令の例は、コンパイラによって生成されるようなマシンコード、およびインタプリタを使ってコンピュータによって実行できる高水準コードを含むファイルを含む。

【００７３】前述の発明は理解を明晰にする目的のためにある程度の詳しさで記述されたが、添付された特許請求の範囲の内で、ある程度の変更や変形を行うことが可能であることは明らかである。例えば、識別子や署名のアルゴリズムは、輸出の法令に合わせて、さらに選択されたり、変形されたり、使用を制限されたりする。これは、データファイルの世界的な交換を考慮するために、ネットワーク化されたコンピュータにおいては特に真実となる。

【００７４】加えて、検証の設定を利用するアプレットを実行することに伴われるステップだけでなく、サイトへの接続を行うために伴われるステップもまた、順序変更できる。本発明の精神や範囲を離れることなく、ステップを削除したり追加したりすることが可能である。

【００７５】さらに、いくつかのセキュリティレベルのみが記述されたが、特定のコンピュータシステムの要求に合わせて、セキュリティレベルは大きく変化できることを理解するべきである。したがって、記述された実施例は、例示的であって制限的でないものとしてとらえるべきであり、この発明はここで与えられた詳細に制限されるものではなく、以下の特許請求の範囲および等価物の全範囲によって定義されるべきである。

【００７６】

【発明の効果】以上説明したように、データファイルの真正性を保証しまた検証するための、特にコンピュータネットワークにわたって転送されると意図されるデータファイルについての、より効率的な方法、装置および生産物が提供される。

【図面の簡単な説明】

【図１】　図１は、ネットワーク化されたコンピュータ環境を表す構成図である。

【図２】　図２は、図１のネットワーク化されたコンピュータ環境で使われる典型的なコンピュータシステムを表わっす構成図である。

【図３】　図３（ａ）は、本発明の実施例で用いられる、アーカイブのデータ構造の実施例であって、署名ファイルが含まれている構成図である。図３（ｂ）は、本発明の実施例で用いられる、署名ファイルの実施例を表わす構成図である。

【図４】　図４は、署名ファイルを持ったデータ構造で使われる、本発明の実施例のフローチャートである。

【図５】　図５は、本発明の実施例に合わせて、セキュリティマネージャの中にセキュリティレベルをセッティングすることに関するステップを表わすフローチャートである。

【図６】　図６は、本発明の実施例に合わせた、高度セッティングを表わすブラウザインターフェースをダイアグラムで表現した構成図である。

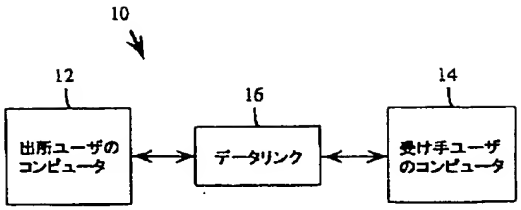
【図７】　図７は、本発明の実施例に合わせた、検証のセッティングを用いるアプレットの実行に関するステップを表わすフローチャートである。

【図８】　図８は、本発明の実施例に合わせた、コンピュータネットワークを介した接続の確立に関するステップを表わすフローチャートである。

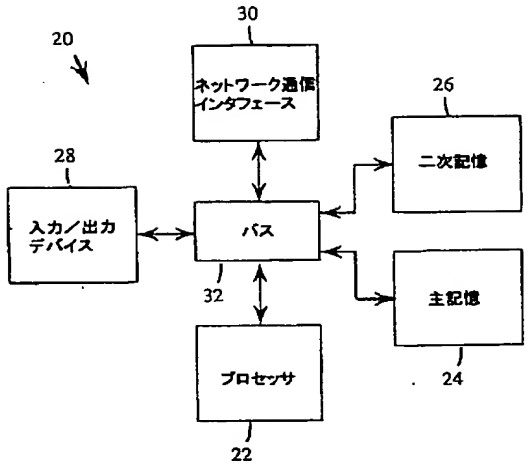
【符号の説明】

１０…計算環境、１２…送り手ユーザのコンピュータ、１６…データリンク、１４…受け手ユーザのコンピュータ、２０…コンピュータシステム、２２…プロセッサ、２４…主記憶、２６…２次記憶、２８…入出力デバイス、３０…ネットワーク通信インターフェース、３２…バス、３０２…署名ファイル、３０４…ファイル１、３０６…ファイル２、３１４…ファイルｎ、３２０…識別子ＩＤ、３２２…署名

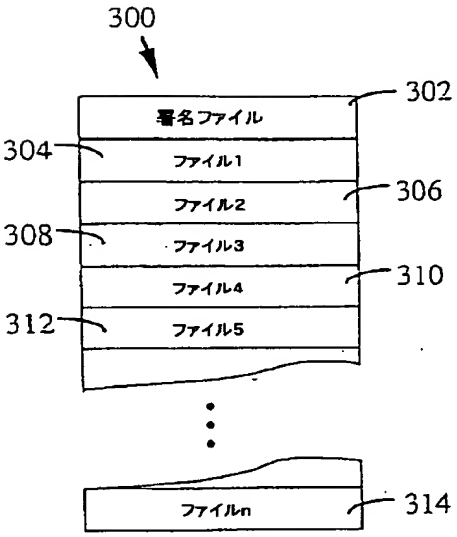
【図１】


10
12
出所ユーザの
コンピュータ
16
データリンク
14
受け手ユーザ
のコンピュータ

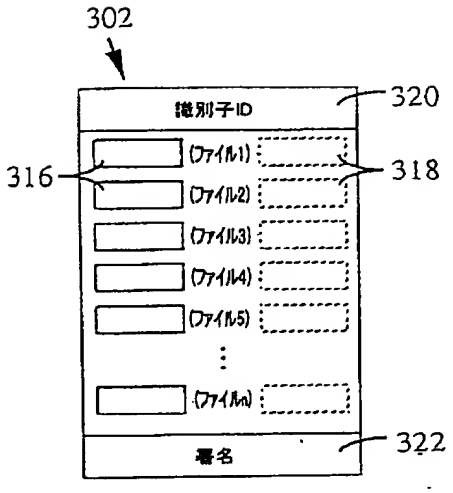
【図２】


20
30
ネットワーク通信
インタフェース
26
二次記憶
28
入力／出力
デバイス
バス
32
主記憶
プロセッサ
24
22

【図３】

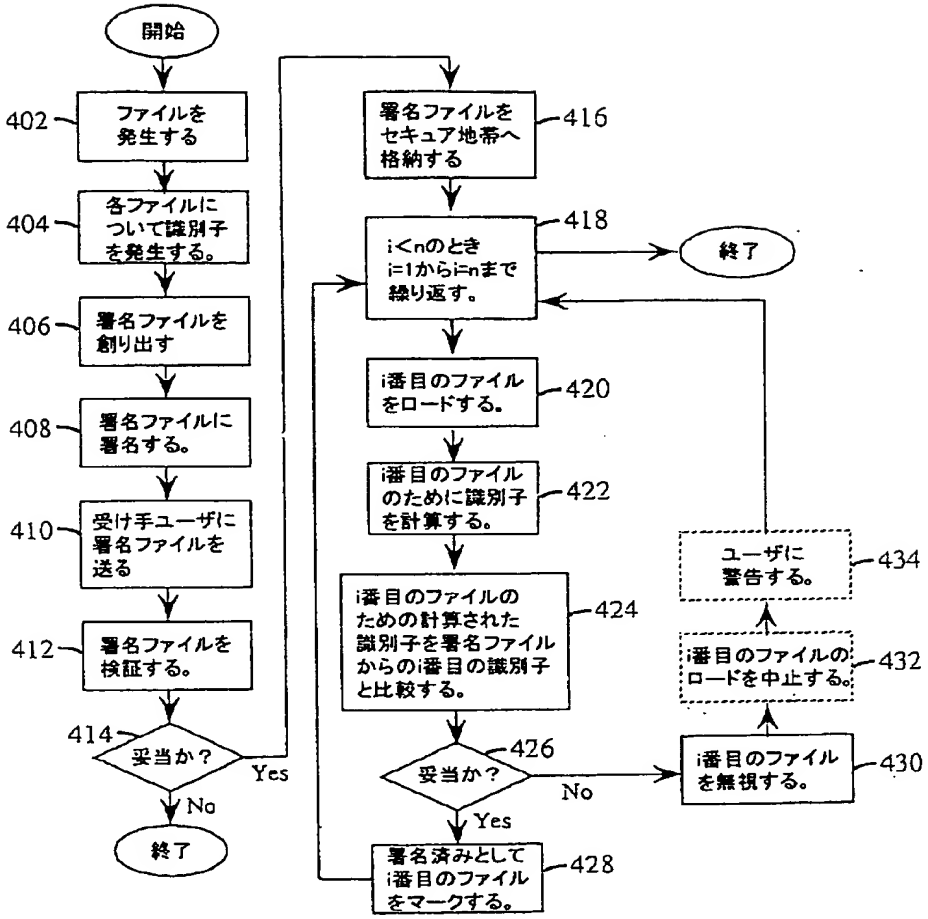

300
署名ファイル
302
304
ファイル1
ファイル2
306
308
ファイル3
ファイル4
310
312
ファイル5
ファイルn
314

(a)

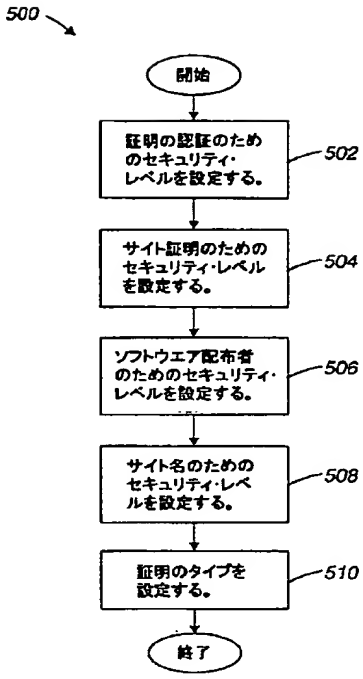

302
識別子ID
320
316
(ファイル1)
(ファイル2)
(ファイル3)
(ファイル4)
(ファイル5)
318
(ファイルn)
署名
322

(b)

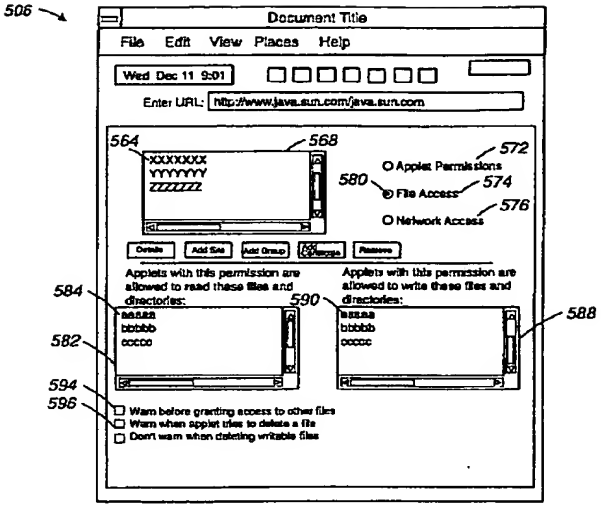
【図４】


開始
402
ファイルを発生する
404
各ファイルについて識別子を発生する。
406
署名ファイルを創り出す
408
署名ファイルに署名する。
410
受け手ユーザに署名ファイルを送る
412
署名ファイルを検証する。
414
妥当か？
Yes
No
終了
416
署名ファイルをセキュア地帯へ格納する
418
i<nのときi=1からi=nまで繰り返す。
終了
420
i番目のファイルをロードする。
422
i番目のファイルのために識別子を計算する。
424
i番目のファイルのための計算された識別子を署名ファイルからのi番目の識別子と比較する。
426
妥当か？
No
Yes
428
署名済みとしてi番目のファイルをマークする。
430
i番目のファイルを無視する。
432
i番目のファイルのロードを中止する。
434
ユーザに警告する。

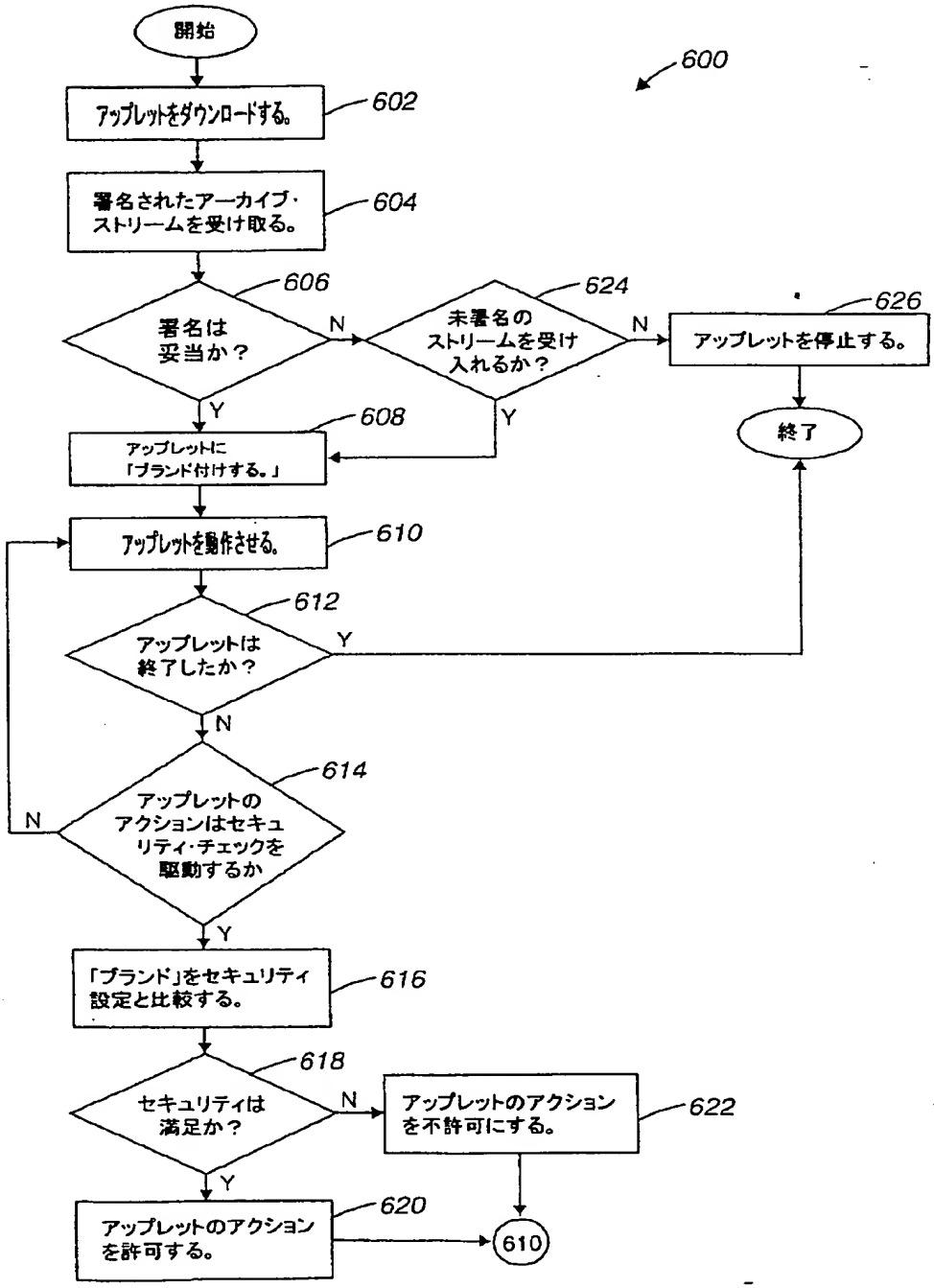
【図５】


500
開始
証明の認証のためのセキュリティ・レベルを設定する。
502
サイト証明のためのセキュリティ・レベルを設定する。
504
ソフトウエア配布者のためのセキュリティ・レベルを設定する。
506
サイト名のためのセキュリティ・レベルを設定する。
508
証明のタイプを設定する。
510
終了

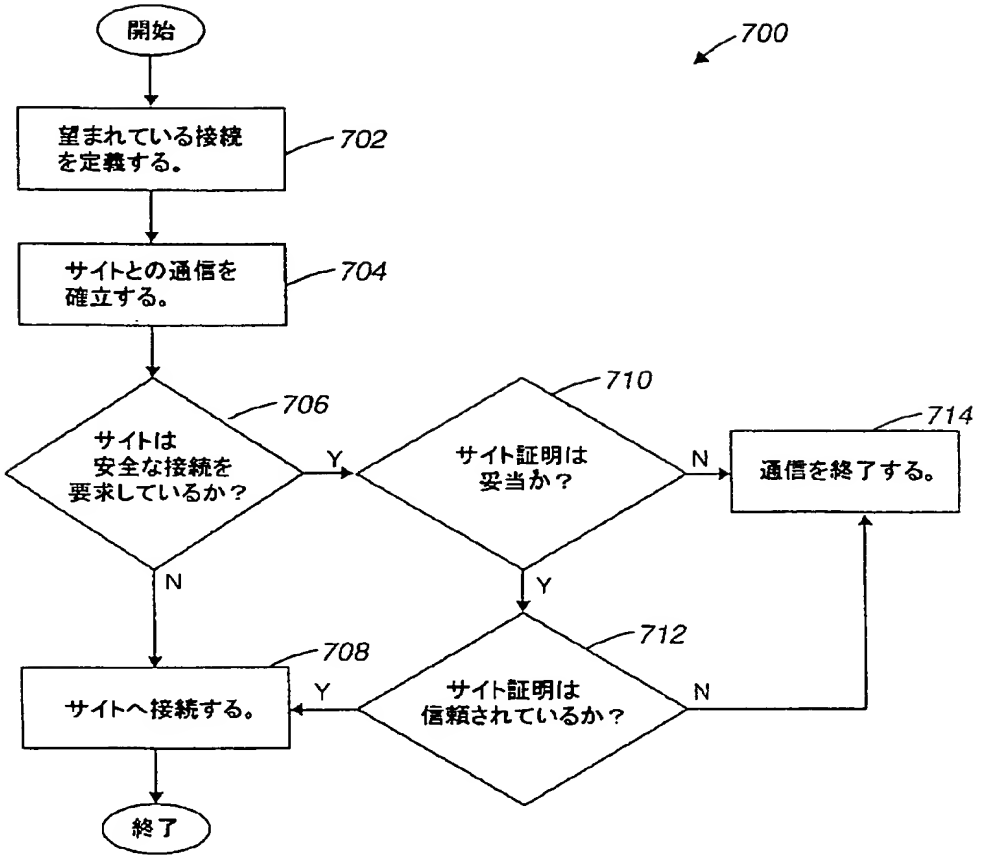
【図６】


506
Document Title
File Edit View Places Help
Wed Dec 11 9:01
Enter URL: http://www.java.sun.com/java.sun.com
564
568
XXXXXXX
YYYYYYY
ZZZZZZZ
572
Applet Permissions
580
File Access
574
576
Network Access
Details
Add Site
Add Group
Remove
584
Applets with this permission are allowed to read these files and directories:
aaaaa
bbbbb
ccccc
590
Applets with this permission are allowed to write these files and directories:
588
582
594
596
Warn before granting access to other files
Warn when applet tries to delete a file
Don't warn when deleting writable files

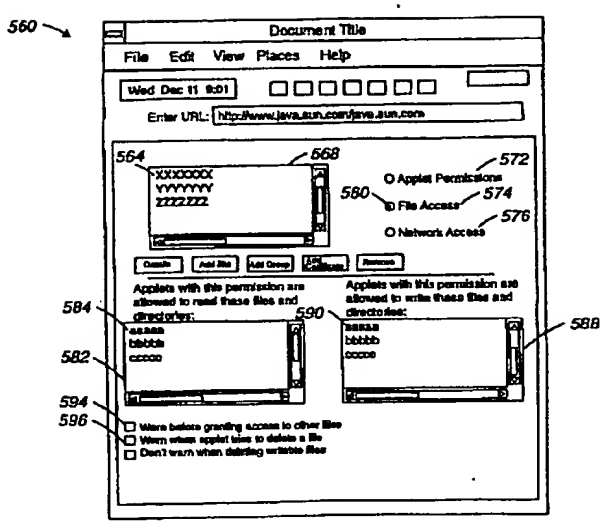
【図７】


開始
600
アップレットをダウンロードする。
602
署名されたアーカイブ・ストリームを受け取る。
604
606
署名は妥当か？
N
624
未署名のストリームを受け入れるか？
N
626
アップレットを停止する。
終了
Y
Y
608
アップレットに「ブランド付けする。」
アップレットを動作させる。
610
612
アップレットは終了したか？
Y
N
614
アップレットのアクションはセキュリティ・チェックを駆動するか
N
Y
「ブランド」をセキュリティ設定と比較する。
616
618
セキュリティは満足か？
N
アップレットのアクションを不許可にする。
622
Y
620
アップレットのアクションを許可する。
610

【図８】



【手続補正書】
【提出日】平成１０年４月９日
【手続補正１】
【補正対象書類名】明細書
【補正対象項目名】請求項４
【補正方法】変更
【補正内容】
【請求項４】　請求項２及び請求項３のいずれか１つに記載の方法であって、
該データファイルの該識別子と該署名ファイルの該識別子とが一致しないとき、該データファイルを無視すると、該データファイルのロードを中止すること、及びユーザに警告すること、の群から選択される少なくとも１つを、更に含む方法。
【手続補正２】
【補正対象書類名】図面
【補正対象項目名】図６
【補正方法】変更
【補正内容】
【図６】



フロントページの続き

(51)Int.Cl.[6] 識別記号
H04L 9/32

F I
H04L 9/00 675Z

(72)発明者 ジヨン シー. バンブッチ
アメリカ合衆国, カリフォルニア州, モーガン ヒル, バレット アヴェニュー 735

(72)発明者 エイブリル イー. ホジーズ ウィルシャー
アメリカ合衆国, カリフォルニア州, パロ アルト, エマーソン ストリート 1085

【外国語明細書】

1. Title of Invention

IMPLEMENTING DIGITAL SIGNATURES FOR DATA STREAMS AND DATA ARCHIVES

2. Claims

What is claimed is:

1. A computer-implemented method for verifying the authenticity of data, the method comprising:

receiving at least one data file and a signature file, wherein the data file and the signature file are separate, the data file including an identifier, the signature file including the identifier for the data file and a digital signature; and

processing the signature file using a computer system to determine the authenticity of the signature file.

2. A computer-implemented method for verifying the authenticity of data as recited in claim 1 further including:

comparing the identifier in the data file with the identifier in the signature file using the computer system to determine the authenticity of the data file, wherein processing the signature file further includes processing the digital signature using the computer system to determine the authenticity of the signature file.

3. The method as recited in claim 2 further including marking the data file as signed when the identifiers in the data and signature files match.

4. The method as recited in one of claims 2 and 3 wherein when the identifiers in the data and signature files do not match, the method further includes at least one selected from the group of ignoring the data file, aborting the loading of the data file, and alerting a user, when the identifiers in the data and signature files do not match.

5. A computer-implemented method for verifying the authenticity of data as recited in one of claims 2-4 wherein comparing the identifier in the data file with the identifier in the signature file using the computer system is repeated for a second data file.

6. A computer-implemented method for verifying the authenticity of data as recited in any one of the preceding claims wherein processing the digital signature further includes verifying the digital signature with a signature algorithm, the signature algorithm being a keyed algorithm, wherein the signature algorithm is selected from a group consisting of a DSA algorithm, and a combined Message Digest and RSA algorithm.

7. A computer-implemented method for verifying the authenticity of data as recited in any one of the preceding claims wherein the identifier is generated using one of a one-way hash function algorithm and a cyclic redundancy checksum algorithm.

8. A computer-implemented method for verifying the authenticity of data as recited in any one of the preceding claims wherein comparing the identifier in the data file with the identifier in the signature file further includes generating one or more of the identifiers with a one-way hash function algorithm.

9. A computer-implemented method for verifying the authenticity of data as recited in any one of the preceding claims wherein comparing the identifier in the data file with the identifier in the signature file further includes checking one or more of the identifiers with a cyclic redundancy checksum algorithm.

10. A computer-implemented method for verifying the authenticity of data as recited in any one of the preceding claims wherein receiving the data file and the signature file further includes transferring the data file and the signature file among networked computers.

11. A computer-implemented method for verifying the authenticity of data as recited in any one of the preceding claims wherein:

the identifier in the data file includes at least one of a certificate authority, a site certificate, a software publisher identifier, and a site name; and

the method includes setting a security level for at least one of said certificate authority, said site certificate, said software publisher identifier, and said site name.

12. A computer-implemented method for verifying the authenticity of data as recited in claim 11 including downloading the data file to the computer system, and when the data file comprises an applet and when the digital signature is verified, the method includes branding the applet as verified and running the applet.

13. A computer-implemented method for verifying the authenticity of data as recited in claim 12 wherein when the data file comprises an applet, and when the signature is not verified, the method includes determining whether an unsigned data file is acceptable for execution on the computer, and terminating the applet if an unsigned data file is not acceptable for execution on said computer.

14. A computer-implemented method for verifying the authenticity of data as recited in claim 13 including branding the applet when the unsigned data file is determined acceptable for execution on said computer.

15. A computer-implemented method for verifying the authenticity of data as recited in claim 14 including the running the applet and determining whether the applet performs an action that triggers a security check.wherein the security check includes comparing the

brand with the security level and allowing the action when the security check is satisfied and disallowing the action if the security check is not satisfied.

16. A computer-implemented method for verifying the authenticity of data as recited in any one of the preceding claims, further including establishing a data communication connection with a remote site using the computer system, determining whether the site requires a secure connection, and determining whether a site certificate for the site is valid in response to a determination that a secure connection is required.

17. An apparatus for verifying the authenticity of at least one data file and a signature file, the data file including an identifier, the signature file including the identifier for the data file and a digital signature, the apparatus comprising:

a processor for processing the digital signature to determine the authenticity of the signature file; and

a comparator for comparing the identifier in the data file with the identifier in the signature file using the computer system to determine the authenticity of the data file, wherein the processor is further arranged to process the digital signature using the computer system to determine the authenticity of the signature file.

18. An apparatus for verifying the authenticity of data as recited in claim 17 wherein the comparator for comparing the identifier in the data file with the identifier in the signature file using the computer system further includes a marker for marking the data file as signed when the identifiers in the data and signature files match.

19. A computer program product including a computer-usable medium having computer-readable program code embodied thereon for use in verifying the authenticity of data, the computer program product including computer-readable program code for effecting the following with a computer system:

a) receiving at least one data file and a signature file, the data file including an identifier, the signature file including the identifier for the data file and a digital signature; and

b) processing the signature file using a computer system to determine the authenticity of the signature file.

20. A computer program product as recited in claim 19 further including computer-readable program code for:

comparing the identifier in the data file with the identifier in the signature file using the computer system to determine the authenticity of the data file, wherein processing the

signature file further includes processing the digital signature using the computer system to determine the authenticity of the signature file.

21. A computer program product as recited in claim 20 further including computer-readable program code for:

comparing the identifier in the data file with the identifier in the signature file using the computer system further includes marking the data file as signed when the identifiers in the data and signature files match.

3. Detailed Description of Invention

FIELD OF THE INVENTION

The present invention relates generally to the sharing of data among computing resources. More specifically, the present invention relates to methods, apparatuses and products for securing and verifying the authenticity of data being processed on a computer system.

BACKGROUND OF THE INVENTION

With the increasing popularity of networked computing environments, such as the Internet, there has been a corresponding increase in the demand to provide for the transferring of shared information among the networked computers in a secure manner. For example, when a user of the Internet sends information in the form of data to another user it may be useful for the receiving user to verify that the data received has not been corrupted or otherwise altered in some manner. Furthermore, the receiving user may also find it useful to verify that the data received was actually sent by the proper sending user rather than an impostor.

As a result, methods and algorithms that increase the security of data transmitted over computer networks and other data links have been developed and deployed with some success. The more secure methods tend to include encrypting all or part of the data prior to sending it, and likewise decrypting the received data prior to using it. Such encryption and decryption techniques may, for example, include adding encryption data to the data file, and encoding or otherwise transforming the data in the data file with a computer system by running a "signature algorithm".

There are currently several signature algorithms in use today. One popular signature algorithm is actually a combination of a Message Digest algorithm and an RSA encryption algorithm (e.g., MD5 with RSA, or MD2 with RSA, or the like). The Message Digest with RSA signature algorithm is available from RSA Data Security, Inc. of Redwood City, CA. Another popular signature algorithm is the DSA encryption algorithm. The DSA encryption algorithm, which is available from the United States Government, may be used for limited purposes by private parties as a signature algorithm. These signature algorithms will be discussed in limited detail below.

The Message Digest with RSA algorithm includes the capability to generate a "digital signature" that can be added to data files. Digital signatures are basically mechanisms through which users may authenticate the source of a received data file. A digital signature is typically a special sequence of data that can be generated and provided along with a related data file to other users. The basic concept behind most signature algorithms is that every user (e.g., individuals, companies, governments, etc.) will have a "key pair" that includes both a "private key" and a "public key". A key may, for example, be a numerical sequence. The private key is a unique key that is assigned to a single user and intended to be kept secret by that user. The private key may be used by the assigned user to create a digital signature for a data file with a signature algorithm. The public key, on the other hand, is typically made available to all other users. The public key may be used by these other users to verify that the digital signature on a received data file is authentic (i.e., that the digital signature was created with the private key). The verification process is accomplished with the same signature algorithm. In principle, such a verification process may provide a relatively high level of confidence in the authenticity of the source of the received data.

In addition to digital signature generating algorithms, there are also algorithms that may be used to authenticate that the data file has not been corrupted in some manner. These algorithms are typically known as "one-way hash functions". One example of such an algorithm is the Message Digest, introduced above. A one-way hash function usually does not require a key. One-way hash functions typically include additional data that is inserted into the data file. As such, when the data file is received the hash function may be used to verify that none of the data within the data file has been altered since the generation of the hash function. However, hash functions are typically limited in that the user may not necessarily infer anything about the associated file, such as who sent it. It is noted that many signature algorithms use one-way hash functions as internal building blocks.

For relatively open, unsecured networks such as the Internet, it may be useful for users to authenticate received data files prior to using them as intended. Such data files may include, but are not limited to, computer programs, graphics, text, photographs, audio, video, or other information that is suitable for use within a computer system. No matter the type of data file, authentication may be accomplished with a signature algorithm or similar type of encryption algorithm as described above. By way of example, if the data file is a software program the user may wish to authenticate that it was sent by a trustworthy authority prior to exposing his or her computer system to the software program, lest the program include a "Trojan Horse" that infects the user's computer with a virus. In such a case, the sending user may authenticate the data as described above.

Another example is where the receiving user wishes to authenticate a text and/or image data file prior to displaying it on his or her computer screen. This may be useful to control the display of text and images having undesirable content. For example, parents may want to limit any access their children may have to pictures and text relating to adult subjects and materials. This can be accomplished by verifying that the data file (e.g., a text or image file), came from a trusted source. Similarly, providers of text and image files may want to provide a "stamp" of approval or authenticity so as to control the use of tradenames and other intellectual property.

Unfortunately, the process of encrypting and decrypting, signing and verifying, and/or generating hash functions places an additional burden on the sending and receiving user's computational resources. The burden is compounded for users who send and receive several data files. By way of example, the growth of that aspect of the Internet known as the World-Wide Web has lead to a tremendous increase in the transfer of multiple data files between users. These multiple data files often include the components or objects that constitute an object-oriented software process, such as a Java™ applet. To illustrate the potential burden that can be placed on the receiving user's computer resources in such a multiple data file transfer, one need only calculate the resulting processing time associated with verifying the digital signatures for each of the files. For example, if an Java™ applet included 200 digitally signed Java™ class files (including data files), assuming that the average verification process took about 1 second on a conventional desktop personal computer, then the user would have to wait for about 200 seconds after receiving the data files to use the applet. Such delays may significantly reduce the effectiveness of such a computer network environment. This is especially true for data files relating to a timed process, such as streaming audio or video data file in real (or near-real) time.

Therefore, what is desired are more efficient methods, apparatuses and products for securing and verifying the authenticity of data files, especially for data files intended to be transferred over computer networks.

SUMMARY OF THE INVENTION

The present invention provides more efficient methods, apparatuses and products for securing and verifying the authenticity of data files, such as data files intended to be transferred over computer networks. In accordance with one aspect of the present invention, a method for verifying the authenticity of data involves providing at least one data file which includes an identifier and a signature file which includes the identifier for the data file as well as a digital signature. The digital signature is then verified using a

computer system, and the identifier in the data file is compared with the identifier in the signature file using the computer system.

In one embodiment, the identifier for the data file includes at least one certificate authority, site certificate, software publisher identifier, or a site name, and verifying the authenticity of data involves setting a security level for at least one of the certificate authority, said site certificate, said software publisher identifier, and said site name. In such an embodiment, the data file is downloaded to the computer system, and if the data file is an applet and the digital signature is verified, then verifying the authenticity of data also involves branding and running the applet accordingly.

In another aspect of the present invention, an apparatus for verifying the authenticity of at least one data file, which includes an identifier, and a signature file which includes the identifier for the data file in addition to a digital signature, includes a verifier for verifying the digital signature and a comparator for comparing the identifier in the data file with the identifier in the signature file. In one embodiment, the digital signature is verified with a signature algorithm. In another embodiment, the comparator includes a one-way hash function algorithm.

In yet another aspect of the present invention, a computer system arranged to verify the authenticity of a data file, which includes an identifier and is associated with a signature file that has the identifier for the data file and a digital signature, includes a processor, a memory coupled to the processor, and a verifier arranged to verify the digital signature and compare the identifier in the data file with the identifier in the signature file. In one embodiment, the identifier for the data file includes at least one of a certificate authority, a site certificate, a software publisher identifier, and a site name. In such an embodiment, the verifier is further arranged to set a security level for at least one of the certificate authority, the site certificate, the software publisher identifier, and the site name. In another embodiment, the data file is an applet and the verifier is arranged both to brand the applet and to run the applet.

The invention, together with further advantages thereof, may best be understood by reference to the following description taken in conjunction with the accompanying drawings in which:

DETAILED DESCRIPTION OF THE INVENTION

Several embodiments of the present invention provide novel methods, apparatuses and products that reduce the computational demands placed on both source user computer systems and receiving user computer systems by requiring the implementation and the verification of only a single digital signature for an arbitrary number of data files. In accordance with an embodiment of the present invention the data files need not be individually signed. Instead, a separate signature file is created such that when the separate signature file is digitally signed and later verified, the data files to which it corresponds can be authenticated without running the signature algorithm for each of these data files. In one embodiment, the signature file includes a list of "identifiers", such as one-way hash functions, that are associated with each of the data files to be transferred. As such, the signature file is essentially the cryptographic equivalent of a digital signature for each of the data files.

Thus, with an embodiment of the present invention a user can create a signature file that includes unique identifiers for each data file. The signature file can be digitally signed by using a signature algorithm. The signed signature file and data files can then be sent to a receiving user, who can then verify the digital signature using the appropriate signature algorithm. Once the digital signature has been verified, the identifiers within the signature file can be compared to the identifiers within the data files. If the identifier within a given data file matches the corresponding identifier in the signature file, then the data file can be verified as being authentic. The receiving user can then proceed to process the verified data files with confidence in their authenticity. As a result, computational delays can be reduced because there is no longer the need to digitally sign and later verify the digital signature for each of the data files.

Figure 1 illustrates a networked computing environment 10, as represented by a block diagram of a source user computer system 12 coupled to exchange information in the form of data with a receiver user computer system 14 over a data link 16. Source user computer system 12 can, for example, take the form of a server computer such as a web server associated with the Internet. Likewise, receiving user computer system 14 can, for example, take the form of a client system that is networked via data link 16 to a web server. In such a case, data link 16 can therefore represent a portion of, or the entire, Internet and other connected networks. Data link 16 can also represent one or more local area networks (LANs), wide area networks (WANs), "intranets" or "extranets", or other like telecommunication or data networks.

Figure 2 illustrates a typical computer system 20 that can be used by either a sending user or a receiving user, in accordance with Figure 1. Alternatively, computer system 20 can be a stand-alone computer capable of receiving data through computer useable products. Computer system 20 includes one or more processors 22, a primary memory 24, a secondary memory 26, one or more input/output (I/O) devices 28, one or more network communication devices 30, and one or more buses 32.

Processors 22 provide the capability to execute computer instructions. Processors 22 can, for example, be microprocessors, central processing units (CPUs),or microcontrollers such as found in many of the desktop, laptop, workstation, and mainframe computers available on the market. Processors 22 can also take the form of conventional or even customized or semi-customized processors such as those typically used in special purpose or larger frame computers, telecommunication switching nodes, or other networked computing devices. Processors 22 are coupled to output data to buses 32 and to input data from buses 32.

Buses 32 are capable of transmitting or otherwise moving data between two or more nodes. Buses 32 can, for example, take the form of a shared general purpose bus or can be dedicated to transmitting specific types of data between specific nodes. Buses 32 can include interface circuitry and software for use in establishing a path between nodes over which data can be transmitted. It is recognized that some devices, such as processors 22 can also include one or more buses 32 internally for transmitting data between internal nodes therein. Data can include processed data, addresses, and control signals.

Primary memory 24 typically provides for the storage and retrieval of data. Primary memory 24 can, for example, be a random access memory (RAM) or like circuit. Primary memory 24 can be accessed by other devices or circuits, such as processors 22, via buses 32.

Secondary memory 26 typically provides for additional storage and retrieval of data. Secondary memory 26 can, for example, take the form of a magnetic disk drive, a magnetic tape drive, an optically readable device such as a CD ROMs, a semiconductor memory such as PCMCIA card, or like device. Secondary memory 26 can be accessed by other devices or circuits, such as processors 22, via buses 32. Secondary memory 26 can, for example, access or read data from a computer program product including a computer-usable medium having computer-readable program code embodied thereon.

I/O devices 28 typically provide an interface to a user through which data can be shared. I/O devices 28 can, for example, take the form of a keyboard, a tablet and stylus, a voice or handwriting recognizer, or some other well-known input device such as, of

course, another computer. I/O devices 28 can also, for example, take the form of a display monitor, flat panel display, or a printer. I/O devices 28 can be accessed by other devices or circuits, such as processors 22, via buses 32.

Network communication devices 30 typically provide an interface to other computing resources and devices, such as other computer systems. Network communication devices 30 typically include interface hardware and software for implementing data communication standards and protocols over data communication links and networks. For example, with a network connection, processors 22 can send and receive data (i.e., information) over a network. The above-described devices and processes will be familiar to those of skill in the computer hardware and software arts.

Figure 3a illustrates an embodiment of an archival data structure 300 in accordance with an embodiment of the present invention. Data structure 300 includes a signature file 302 and several data files 304-314. Files 304-314 can be any digital bit stream, such as, for example, Java™ class files, image files, audio files, text files, and even additional signature files.

Figure 3b illustrates an embodiment of a signature file 302. It should be appreciated that in some embodiments, signature file 302 is a header file. In the illustrated embodiment, signature file 302 includes at least one identifier 316 for each of the data files 304-314. Optionally, signature file 302 can also contain additional data 318 for each of the data files 304-314. For example, additional data 318 can further comprise the information about the name of the file, the author of the file, the date of the file, the version of the file, the file's rating (e.g., movie rating, such as "PG"), or any other authenticated data that the users may want to include within signature file 302.

Signature file 302 further includes an identifier ID 320 and a digital signature 322. Identifier ID 320 provides the information necessary to determine the algorithm(s) used to create the identifiers listed in signature file 302. Digital signature 322 represents the digital signature created for the signature file. The structure of digital signature 322 will depend, of course, on the signature algorithm used to create it.

Figure 4 illustrates a method 400, in accordance with an embodiment of the present invention, that includes step 402 for generating one or more data files. Step 402 can, for example, include using a text program to generate a text file, a recording program to generate an audio or video file, a graphics program to generate an image or movie file, a programming language to generate a class file or program file, or any other mechanism that is capable of generating a data file.

Having generated one or more data files in step 402, step 404 includes generating an identifier for each of these data files. The identifiers generated in step 404 can, for example, be generated by a one-way hash function algorithm, or alternatively can even take the form of a cyclic redundancy checksum (CRC), or the like. It is recognized, however, that generally a one-way hash function algorithm tends to provide for greater security because such functions cannot be easily or efficiently broken, or otherwise reverse-engineered. By way of example, one-way hash function algorithms, such as MD5 and SHA are typically considered to be cryptographically secure. Such algorithms will be known to those having skill in the computer science art.

Next, step 406 includes creating a signature file that lists, or otherwise compiles, the identifiers as generated in step 404. A signature file can, for example, be a text file that lists the identifiers. Optionally, a signature file can further include, for example, the name of each file, the author of each file, the file version, a date-stamp for the file, or other data relating to each data file. Step 406 can further include one or more programs that inquire, trace, select, or otherwise gather or render such data from the data files. Step 406 can be performed, for example, by processing the data files in a batch mode process to gather the appropriate identifiers and any additional data. Those skilled in the art will recognize that there can be benefits (e.g., in efficiency) to specifically ordering, grouping or otherwise arranging the data listed in the signature file in some manner that expedites the steps in method 400. For example, it can be useful to group the file name or the author along with the identifier.

Once the signature file has been created, step 408 includes digitally signing the signature file with a signature algorithm. Examples of suitable signature algorithms include a combined Message Digest algorithm and RSA encryption algorithm (e.g., MD5 with RSA, or MD2 with RSA, or the like), or the DSA algorithm (discussed above). Step 408 can also include, for example, generating a digital signature for the signature file with a signature algorithm by way of a public or private key (e.g., see Schneier, above).

The signed signature file from step 408 is then sent, provided or otherwise made available to the receiving user in step 410. Step 410 can, for example, include transmitting the signed signature file over a data bus, data link, the Internet, or some other computer or data communication network or link. In addition it is recognized that step 410 can, for example, include storing the signature file in a computer readable medium like a magnetic storage media or optical storage media, and moving the signed signature file on the computer readable medium from one computer to another computer.

Upon receipt or access, the receiving user in step 412, verifies the authenticity of the signed signature file sent or made available in step 410. Step 412 can, for example, include

verifying the digital signature on the signed signature file with a signature algorithm by way of a key.

Step 414 represents a decision wherein the validity of the digital signature as determined in step 412 either terminates or continues method 400. While depicted as interrupting or otherwise preempting method 400, it is recognized that step 414 can also include invoking another process, such as an alarm or notification process, or log process, that in some way records or identifies, or otherwise addresses that that verification of the signed signature file in step 412 failed.

If the decision in step 414 is that the file is valid (i.e., authentic), then the process continues to step 416 which includes storing at least the identifiers from the signature file. In one embodiment of the present invention, the identifiers are stored in a secure location. A secure location can, for example, be the RAM of the receiving computer system since this memory is readily cleared when the process is completed. Alternatively, the identifiers can be stored to a disk or tape drive wherein they can be retrieved at some later stage. Those skilled in the art will recognize that various data storage devices and other computer system configurations pose varying and potential security risks (i.e., some storage devices will be more secure than others). It is also recognized that additional security measures, such as encryption and file access privileges, can be used to further secure or increase the trustworthiness of the signature file as stored in step 414.

Once the identifiers have been stored in a secure location in step 416, then the data file or data files whose identifiers were listed in the signature file in step 406 can then be processed in accord with a loop as represented in step 418. Step 418 can, for example, include a counter mechanism that iteratively controls the number of times that step 420 will be entered into based on the number of identifiers listed in the signature file. For example, if there are "n" number identifiers listed in the signature file (i.e., there are n data files to be loaded), then an iterative loop can count up from i = 1 to i = n, or alternatively down from i = n to i = 1, or otherwise determines when all of the data files have been loaded, or that loading has been attempted, in accord with the remainder of the steps in method 400 as presented below.

Step 420 includes loading the i^{th} data file. Step 420 can, for example, include any of the methods in step 410 to either download, upload, broadcast, or otherwise move the i^{th} data file from one location to another location. Once the i^{th} data file has been loaded, step 422 includes providing, computing or generating the identifier for the i^{th} data file with the appropriate identifier algorithm (for that data file).

Next, step 424 includes comparing the identifier provided in step 422 with the identifier listed, for the i^{th} data file, in the signature file which was stored in step 416. If the identifiers match then the i^{th} data file is verified as authentic. If the identifiers do not match then the i^{th} data file is considered not to have been verified.

Step 426 represents a decision wherein the validity of the identifier, as determined in step 424, either interrupts or continues the iterative loop of step 418. If the identifier for the i^{th} data file has been verified in step 424, then step 426 continues the iterative loop of step 418 by proceeding to step 428 which includes marking, or otherwise recording or establishing in some manner, that the i^{th} data file has been verified as being authentic. Step 428 can, for example, include modifying or marking the i^{th} data file as having been signed by the source user.

If, on the other hand, the identifier for the i^{th} data file is not verified as authentic in step 424, then step 426 interrupts the iterative loop of step 418 by proceeding to step 430. Step 430 includes interrupting the iterative loop of step 418 in some manner so as to avoid step 428 and to proceed back to step 418. Step 430 can, for example, include ignoring the i^{th} data file. In addition to step 430, other steps can be included in method 400 to somehow record or otherwise identify that the i^{th} data file is not authentic.

Thus, with the data structure and steps above, a user who is sending several data files will likely reduce the associated processing time because rather than having to generate a separate digital signature for each data file, the sending user need only create a signature file and digitally sign that file. Likewise, with the data structure and steps above, the user who is receiving several data files will likely reduce the associated processing time because rather having to verify each data file as being authentic by decrypting an associated digital signature, the receiving user need only verify that the signature file is authentic. Such a hybrid verification process substantially streamlines the signature and verification processes. As a result, data files may be digitally signed, and later authenticated and processed in less time.

Additionally, step 430 can lead to optional step 432 which aborts the attempted load, and/or to optional step 434 which alerts or otherwise warns of the failure to verify authentication in step 424. Once steps 430, and optionally 432 and/or 434, have been completed then method 400 returns to step 418 to complete the iterative loop therein. Once the iterative loop of step 418 has been completed, then method 400 is ended.

In one embodiment of the present invention, for each signatory authority, a certificate is created, i.e., the unique identifiers which are listed in a signature file are embodied as certificates. In general, the certificates are tokens that a site, which is typically

either a source user computer system or a receiving user computer system, can use to identify itself. Several sites can be associated with a single certificate. Alternatively, several certificates can be associated with one site.

A source user computer system and a receiving user computer system can be configured to exchange not only data files but computer software in the form of "applets," such as those written in the Java™ programming language available from Sun Microsystems of Mountain View, California. "Applets" as used herein are software programs that are configured to be passed from a source computer, typically a server, to a client machine and run in conjunction with software, as for example browser software, already installed on the client. In the described embodiment, applets are instantiated from class files, which are grouped together into an archival data structure as described above with respect to Figure 3a, that are downloaded from a source computer, or a server, to a client machine. Typically, applets provide additional functionalities to browser software by performing various computational tasks which the browser software itself is not configured to perform. Thus, users who download applets can provide the browser software with additional functionalities that are not otherwise available to the browser software. Such additional capabilities can include, e.g., custom interfaces to a database.

A security manager associated with a browser can be used on either a source computer or a client machine to control operations which are accessible to given applets, as for example a Java™ applet. In other words, a security manager can be used to control the actions which an applet is allowed to perform, or otherwise extend privileges to applets. Although the actions which an applet is allowed to perform can be widely varied, in general, the actions are read and write actions. Within a security manager, different security levels can be implemented to provide a user with the flexibility to set permissions for different certificates and sites associated with an applet. Generally, a user can select a particular certificate or site, or a group of certificates and sites, and set the security level for his selection.

Using a security manager to implement security levels generally entails identifying which applet actions are considered to be secure, safe, or trusted, as well as applet actions which are considered to be unsecure, unsafe, or not trusted. A secure action is generally an action which is not considered to have serious potential for jeopardizing system security or for corrupting information stored on a client or a server. By way of example, a secure action can be a read-only action, or a read-only action for a particular directory. On the other hand, an unsecure action is generally any action which has potential for violating system security or for damaging information stored on a client or a server. Unsecure actions can include, but are not limited to, writing actions, deleting actions, renaming actions, and even reading actions which request access to sensitive documents. Unsecure

actions can further include requests to establish connections to remote sites which are protected.

In one embodiment of the present invention, a browser which can run Java™ applets, as for example a HotJava™ browser (available from Sun Microsystems of Mountain View, California), has a security manager with security levels which include a high security level, a medium security level, a low security level, and an untrusted level. A high security level essentially enables applets to run with a set of safe actions, or constraints, while blocking any unsafe actions. In the described embodiment, the high security level enables applets to perform most actions which are considered to be safe, e.g., trusted, while denying access for any actions which are considered to be unsafe, e.g., not trusted.

A medium security level can be used to enable applets to run with safe constraints, while providing users with the ability to grant permissions for actions which can potentially be unsafe. With a medium security level, a user can be warned through a user interface of an action which may not be a safe, e.g., allowable, action. In the described embodiment, a dialog box which describes the activity appears, and the user isprompted to either grant or deny permission for the potentially unsafe action to be executed. A low security level allows applets to run with minimal constraints, and in the described embodiment, does not warn the user of potentially unsafe actions. An untrusted security level is used to identify certificates and sites which are known to be unsafe.

Referring next to Figure 5, the steps associated with setting security levels in a security manager will be described in accordance with one embodiment of the present invention. In one embodiment, the security levels, also known as levels of trust and verification settings, are the high security level, the medium security level, the low security level, and the untrusted security level, as previously described. Although the security levels can be set by a user through the use of a suitable graphical user interface (GUI), it should be appreciated that any suitable method can be used to set security levels.

The process of setting security levels in a security manager 500 begins, and in step 502, security levels for the certificate authority are set. The certificate authority enables different security levels, or priorities, to be applied to both individual certificates and groups of certificates. In general, the certificate authority can include, but is not limited to, information which identifies how a particular certificate is to be used. By way of example, a certificate authority can be set to enable one certificate to "vouch" for, or authenticate, other certificates.

After the security levels for the certificate authority are set, then the security levels for site certificates are set in step 504. Site certificates are certificates which a given site can

use to initiate a secure connection over which a transaction can be made. Security levels for site certificates generally involve specifying secure socket layer (SSL) standard protocols and security permissions which can be used to authenticate connections over which secure transactions are to occur. Such secure communications technologies can be used to identify rogue, or potentially unsecure, sites and, hence, provide more secure channels over which transmissions can be made by avoiding communication with such sites.

In step 506, security levels for software publishers are set. It should be appreciated that in intranet environments, software is typically not published with certificates, as an intranet environment is usually considered to be a secure environment. Hence, software published within such a secure environment is generally assumed to be secure. However, for environments in which software is published with certificates, as for example in internet environments, the certificates can be used to determine if software code associated with the certificates is trusted for browser execution.

Security levels for site names are set in step 508. The process of setting security levels for site names is essentially the same as the process for setting software publishers, except that when a security level is set for a site, the security level is applied to all software associated with the site. Setting site name permissions generally enables software without certificates to be tested with little risk of tampering with system resources. Then, certificate types are set in step 510. Setting certificate types can entail determining how certificates are to be used, and choosing authorities for the certificates based upon how the certificates are expected to be used. After the certificate sites are set in step 510, the process of setting security levels is completed. It should be appreciated that the order in which the security levels are set can be widely varied depending upon the requirements of a particular security manager.

In some embodiments, additional "advanced" settings can be used to set security levels. In one embodiment, advanced settings are granularity controls which a user can modify through a GUI or similar interface enables the user to set specific, customized security levels for individual certificate authorities, site certificates, software publishers, or site names. Further, advanced settings can also be configured to enable a user to customize a security level for a group of certificate authorities, site certificates, software publishers, or site names. Advanced settings thus generally provide for flexibility in controlling security levels and in overall certificate-handling. By way of example, through the use of advanced settings, a user can set a particular site certificate to a medium security level, while also specifying that the security permissions associated with the site certificate be limited to only allowing read access. In addition, advanced settings can also be used to override specifications at given security levels, e.g., advanced settings can be used to grant permissions which are not normally allowed at a given security level.

In the described embodiment, advanced settings are provided such that in addition to the high security level, the medium security level, the low security level, and the untrusted security level which were previously discussed, additional options are available to enable a user to select specific permissions for a site certificate, for example, which can be implemented in addition to selected security levels. These specific permissions can include, but are not limited to, allowing an applet to open windows without providing a warning message to a user, allowing an applet to automatically launch local applications with or without a warning dialog, allowing an applet to access all properties without providing a warning dialog, and allowing an applet to begin execution without providing a warning dialog.

Figure 5a is a diagrammatic representation of a browser interface which illustrates advanced settings in accordance with an embodiment of the present invention. As previously mentioned, advanced settings are used to select specific permissions which can be enabled or disabled in addition to the security levels provided with the browser interface. Although browser interface 560 can be any suitable browser interface 560, in the described embodiment, browser interface 560 is a basic representation of a HotJava™ browser. As shown, browser interface 560 includes an advanced settings display window 564. A first region 568 of display window 564 lists sites and certificates 570, as well as groups of sites and certificates, for which security permissions can be customized. "Applet Permissions" 572, "File Access" 574, and "Network Access" 576 are among permissions which can be customized. In the described embodiment, a selection 580 indicates that permissions for File Access 574, i.e., files which an applet is allowed to access, are to be set.

A second sub-region 582 displays files and directories 584 which, when selected, an applet with the selected settings, i.e., settings determined using the Applet Permissions 572 command, is allowed to read from. Similarly, third sub-region 588 displays files and directories 590 to which an applet with suitable permissions is allowed to write. Additional selectable options, as for example a "Warn before granting access to other files" option 594 or a "Warn when applet tries to delete a file" option 596 can further enable a user to customize security options.

Figure 6 is a flow chart which illustrates the steps associated with one process of executing an applet that uses verification settings in accordance with one embodiment of the present invention. The process of implementing verification settings begins, and in step 602, an applet is downloaded to a local machine on which the applet is to be executed. In the described embodiment, downloading the applet entails downloading at least part of the archive file, or archival data structure, that contains the class files from which the applet can be instantiated. After the applet is downloaded, a signed archive stream is received in step

604. In the described embodiment, the archive stream contains a digital signature that is associated with a Java™ archive file.

In step 606, a determination is made regarding whether the signature in the signed archive stream is valid, i.e., whether the signature is known and acceptable. It should be appreciated that the signed archive stream is one embodiment of the signed signature file which was previously described. The determination of whether the signature in the archive stream is valid can entail systematically checking a chain of authorities, e.g., certificates, in a signature file until a known authority is found. The known authority is then checked for validity. By way of example, a certificate "A" can be vouched for by a certificate "B," which is known to be a valid certificate. Hence, as certificate "B" is known to be valid, certificate "A" can then be assumed to be valid.

If the determination is that the signature is valid, then process flow proceeds to step 608 in which the applet is "branded." Branding, or marking, an applet generally refers to attaching a signer to the applet, or attaching an identifier to the applet which can be used to identify the validity of the applet. Once the applet is suitably branded, the applet is executed, or run, in step 610. While the applet is running, various actions within the applet are called.

A determination is made in step 612 regarding whether the applet has finished running. In other words, it is determined whether each action associated with the applet has either been executed or been disallowed from executing, as will be described below. If it is determined that applet execution has been completed, the process of executing an applet ends. If it is determined that the applet execution has not been completed, then in step 614, it is determined whether the applet action triggers a security check. That is, in step 614, a determination is made regarding whether a particular applet action falls within those actions determined to be potentially harmful to a user's system security. Such actions will be familiar to those of skill in the computer arts, and, more specifically, to those of skill in the computer security arts. By way of example, actions which are potentially harmful to the user's system security can include, but are not limited to, unrestricted write access, modification of system resources, and open transmission to other systems.

If the applet action triggers a security check, then the brand placed on the applet in step 608 is compared with security settings, or permission levels, which were previously provided by a user in step 616. In some embodiments, comparing security settings with the brand on the applet involves a consultation with a user through a user interface. In such embodiments, the user can authorize a bypass of security settings, i.e., the user can override the security settings to either allow or deny a particular action. From step 616, process control proceeds to step 618 which is the determination of whether security is

satisfied for the applet action. If security is satisfied, either by virtue of the fact that the brand placed on the applet compares favorably with the security settings, or by virtue of the fact that the user has authorized the applet action, then the applet action is allowed in step 620. Process control then returns to step 610, and the continued execution of the applet. On the other hand, if security is not satisfied in step 618, then the applet action is disallowed in step 622. After the applet action is disallowed, process control returns to step 610 in which the applet continues to run.

If the applet action does not trigger a security check in step 614, then process flow returns to step 610 in which the applet continues to run. Process flow continues to loop between steps 610 and 614 until either the applet has finished executing, or a determination is made in step 614 that the current applet action triggers a security check, in which case process control proceeds to step 618 as previously described.

Returning to the check for signature validity in step 606, if a determination is made that the signature is not valid, then the archive is considered to be unsigned, and process flow proceeds to step 624 which is the determination of whether the unsigned stream should be accepted. In the described embodiment, the determination of whether the unsigned stream should be accepted is made by a user through the use of a user interface. By way of example, the user can be prompted with a warning dialog which indicates that while the signature was not valid, he can make the decision to run the applet. If the determination is made that the unsigned stream is to be accepted, process flow moves to step 608 in which the applet is branded as appropriate, e.g., branded to indicate that the stream associated with the applet is unsigned. If the determination in step 624 is that the unsigned stream is not to be accepted, then the applet is stopped, or prohibited, from running in step 626, and the process of executing an applet ends.

Referring next to Figure 7, the steps associated with establishing a connection across a computer network will be described in accordance with one embodiment of the present invention. The process of making a connection 700 begins at step 702 in which the desired connection is defined. In the described embodiment, defining the desired connection entails specifying a universal reference language (URL) address for the site to which a connection is desired. After the connection is defined, communication is established with the site to which a connection is desired in step 704.

From step 704, process control proceeds to step 706 in which a determination is made regarding whether the site requires a secure connection. In one embodiment, a secure connection is a connection over a secure socket layer (SSL), as will be appreciated by those skilled in the art. If it is determined that a secure connection is not required, a connection to the site is made in step 708, and the process of establishing a connection is completed.

If it is determined that the site requires a secure connection, then in step 710, it is determined whether the site certificate associated with the site is valid. It should be appreciated that a valid site certificate is not necessarily a trusted site certificate, as a site certificate can be a valid certificate for a site that is not trusted, or is known to be unsafe. If the site certificate is valid, then process flow moves to step 712, which is the determination of whether the site certificate is trusted. If the determination is that the site certificate is trusted, then process flow proceeds to step 708 in which a connection is made to the site. Alternatively, if the determination is that the site certificate is not trusted, then in step 714, the communication which was established with the site (in step 704) is terminated. Similarly, if it is determined in step 710 that the site certificate is not valid, then communication with the site is terminated in step 714.

The embodiments of the present invention as described above employs various process steps involving data stored in computer systems. These steps are those requiring physical manipulation of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It is sometimes convenient, principally for reasons of common usage, to refer to these signals as bits, values, elements, variables, characters, data structures, or the like. It should be remembered, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

Further, the manipulations performed are often referred to in terms such as generating, calculating, computing, marking, ignoring, aborting, alerting, verifying, signing, sending, receiving, creating, iterating, identifying, running, or comparing. In any of the operations described herein that form part of an embodiment of the present invention, these operations are machine operations. Useful machines for performing the operations of an embodiment of the present invention include general purpose digital computers or other similar devices. In all cases, there should be borne in mind the distinction between the method of operations in operating a computer and the method of computation itself. An embodiment of the present invention relates to method steps for operating a computer in processing electrical or other physical signals to generate other desired physical signals.

An embodiment of the present invention also relates to an apparatus for performing these operations. This apparatus may be specially constructed for the required purposes, or it may be a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. The processes presented herein are not inherently related to any particular computer or other apparatus. In particular, various general purpose machines may be used with programs written in accordance with the teachings herein, or it may be more convenient to construct a more specialized apparatus to perform the required

method steps. The required structure for a variety of these machines will appear from the description given above.

In addition, an embodiment of the present invention further relates to computer readable media that include program instructions for performing various computer-implemented operations. The media and program instructions may be those specially designed and constructed for the purposes of an embodiment of the present invention, or they may be of the kind well known and available to those having skill in the computer software arts. Examples of computer-readable media include, but are not limited to, magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROM disks; magneto-optical media such as floptical disks; and hardware devices that are specially arranged to store and perform program instructions, such as read-only memory devices (ROM) and random access memory (RAM). Examples of program instructions include both machine code, such as produced by a compiler, and files containing higher level code that may be executed by the computer using an interpreter.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. For instance, the identifiers or signature algorithms may be further selected, modified or otherwise limited in use so as to adhere to export regulations. This is especially true for computers networked to provide for the global exchange of data files.

In addition, steps involved with establishing a connection to a site, as well as steps involved with executing an applet which uses verification settings, may be reordered. Steps may also be removed or added without departing from the spirit or the scope of the present invention.

Further, although only a few security levels have been specified, it should be appreciated that the security levels can be widely varied in accordance with the requirements of a particular computer system. Therefore, the described embodiments should be taken as illustrative and not restrictive, and the invention should not be limited to the details given herein but should be defined by the following claims and their full scope of equivalents.

4. Brief Description of Drawings

FIGURE 1 illustrates a networked computing environment.

FIGURE 2 illustrates a typical computer system for use with the networked computing environment in Figure 1.

FIGURE 3a illustrates an embodiment of an archival data structure, including a signature file, for use with an embodiment of the present invention.

FIGURE 3b illustrates an embodiment of a signature file, for use with an embodiment of the present invention.

FIGURE 4 is a flow chart of an embodiment of the present invention for use with data structures having signature files.

FIGURE 5 is a flow chart which illustrates the steps associated with setting security levels in a security manager in accordance with an embodiment of the present invention.

FIGURE 5a is a diagrammatic representation of a browser interface which illustrates advanced settings in accordance with an embodiment of the present invention.

FIGURE 6 is a flow chart which illustrates the steps associated executing an applet which uses verification settings in accordance with an embodiment of the present invention.

FIGURE 7 is a flow chart which illustrates the steps associated with establishing a connection across a computer network in accordance with an embodiment of the present invention.

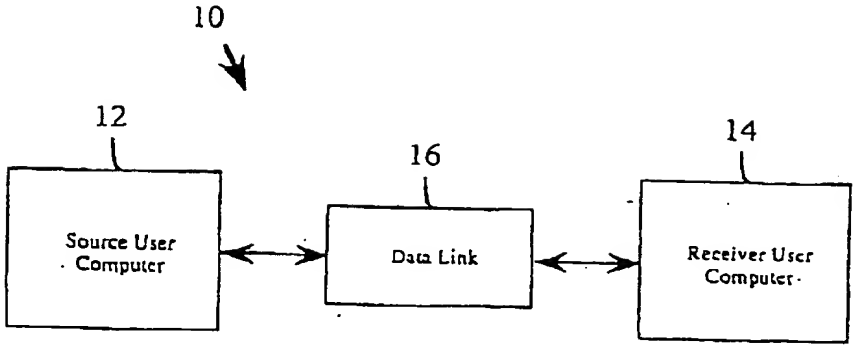

10
12
16
14
Source User Computer
Data Link
Receiver User Computer

FIGURE 1

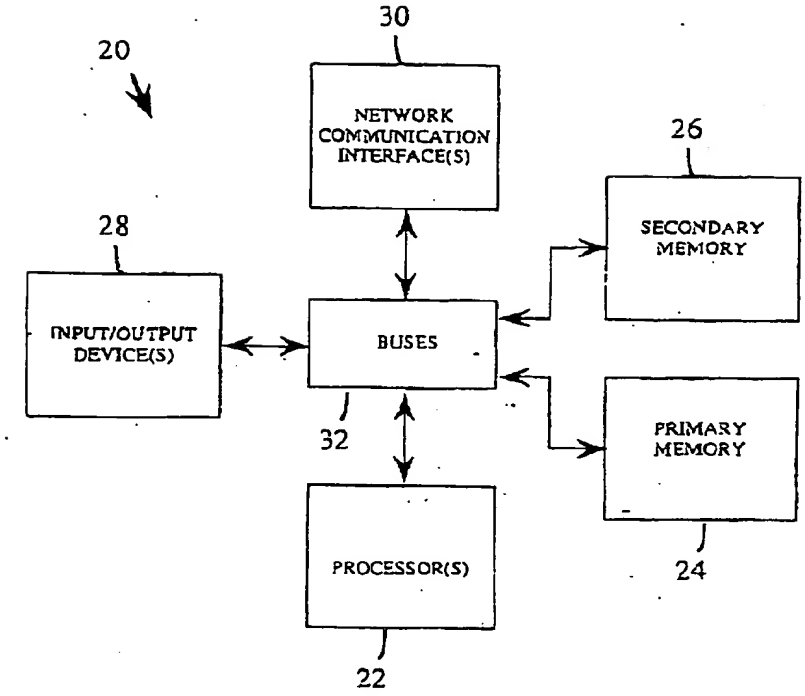

20
30
NETWORK COMMUNICATION INTERFACE(S)
26
SECONDARY MEMORY
28
INPUT/OUTPUT DEVICE(S)
BUSES
32
PRIMARY MEMORY
PROCESSOR(S)
24
22

FIGURE 2

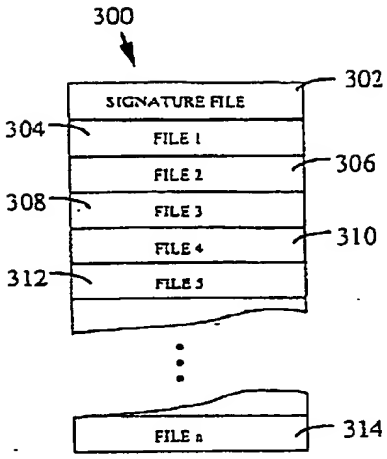


FIGURE 3a

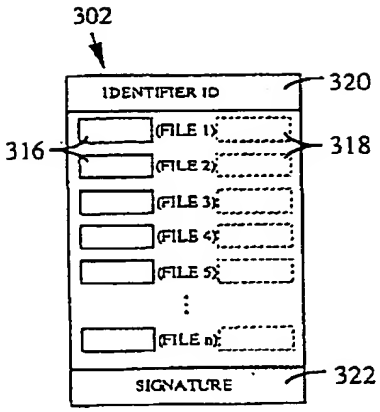


FIGURE 3b

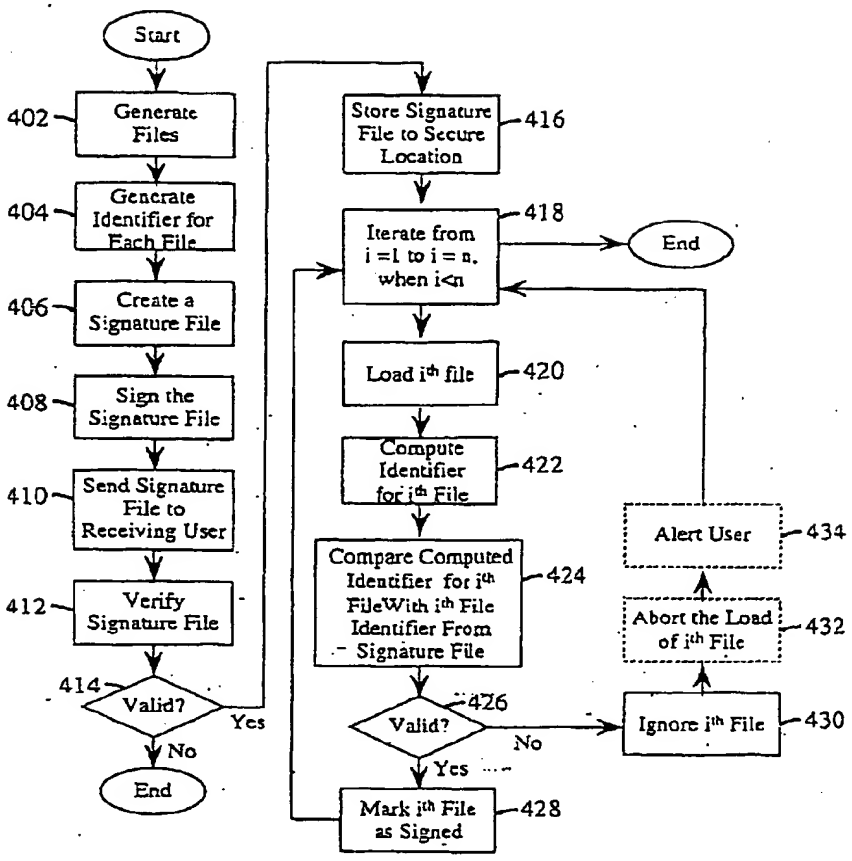

Start
402
Generate Files
404
Generate Identifier for Each File
406
Create a Signature File
408
Sign the Signature File
410
Send Signature File to Receiving User
412
Verify Signature File
414
Valid?
Yes
No
End
Store Signature File to Secure Location
416
Iterate from i =1 to i = n, when i<n
418
End
Load i^th file
420
Compute Identifier for i^th File
422
Compare Computed Identifier for i^th FileWith i^th File Identifier From Signature File
424
426
Valid?
No
Yes
Mark i^th File as Signed
428
Ignore i^th File
430
Abort the Load of i^th File
432
Alert User
434

FIGURE 4

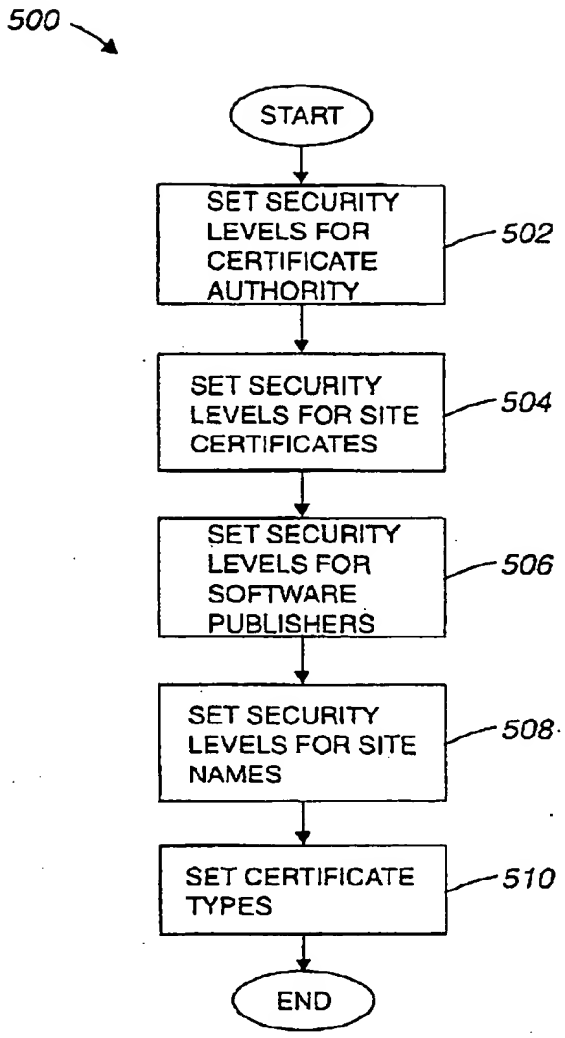

500
START
SET SECURITY LEVELS FOR CERTIFICATE AUTHORITY
502
SET SECURITY LEVELS FOR SITE CERTIFICATES
504
SET SECURITY LEVELS FOR SOFTWARE PUBLISHERS
506
SET SECURITY LEVELS FOR SITE NAMES
508
SET CERTIFICATE TYPES
510
END

Figure 5

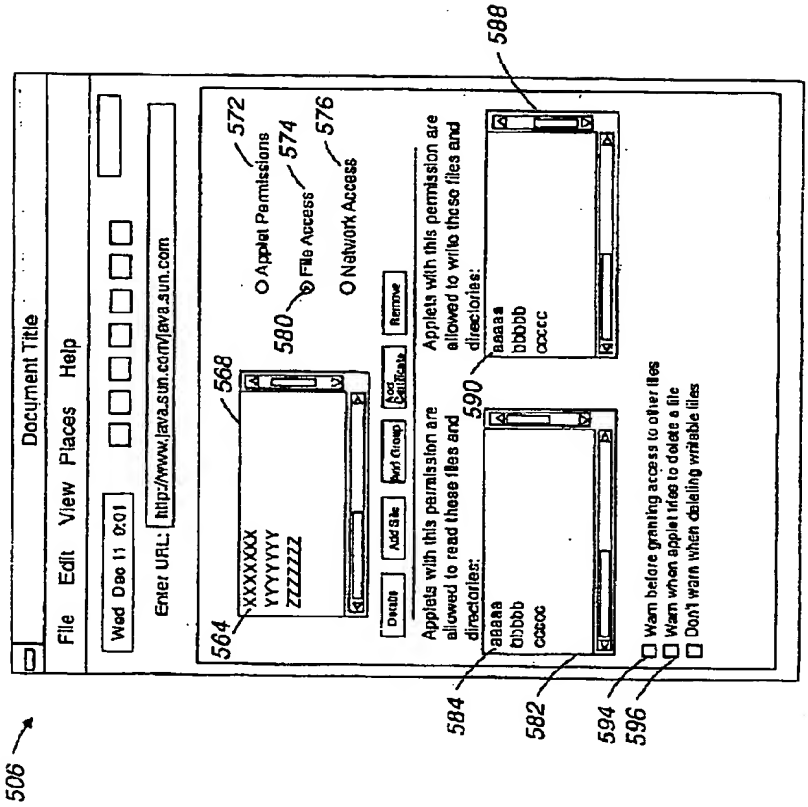

Document Title
File Edit View Places Help
Wed Dec 11 0:01
Enter URL: http://www.java.sun.com/java.sun.com
Applet Permissions
File Access
Network Access
XXXXXXX
YYYYYYY
ZZZZZZZ
Details
Add Site
Add Group
Add Certificate
Remove
Applets with this permission are allowed to read these files and directories:
aaaaa
bbbbb
ccccc
Applets with this permission are allowed to write these files and directories:
aaaaa
bbbbb
ccccc
Warn before granting access to other files
Warn when applet tries to delete a file
Don't warn when deleting writable files
506
564
568
572
574
576
580
582
584
588
590
594
596

Figure 5a

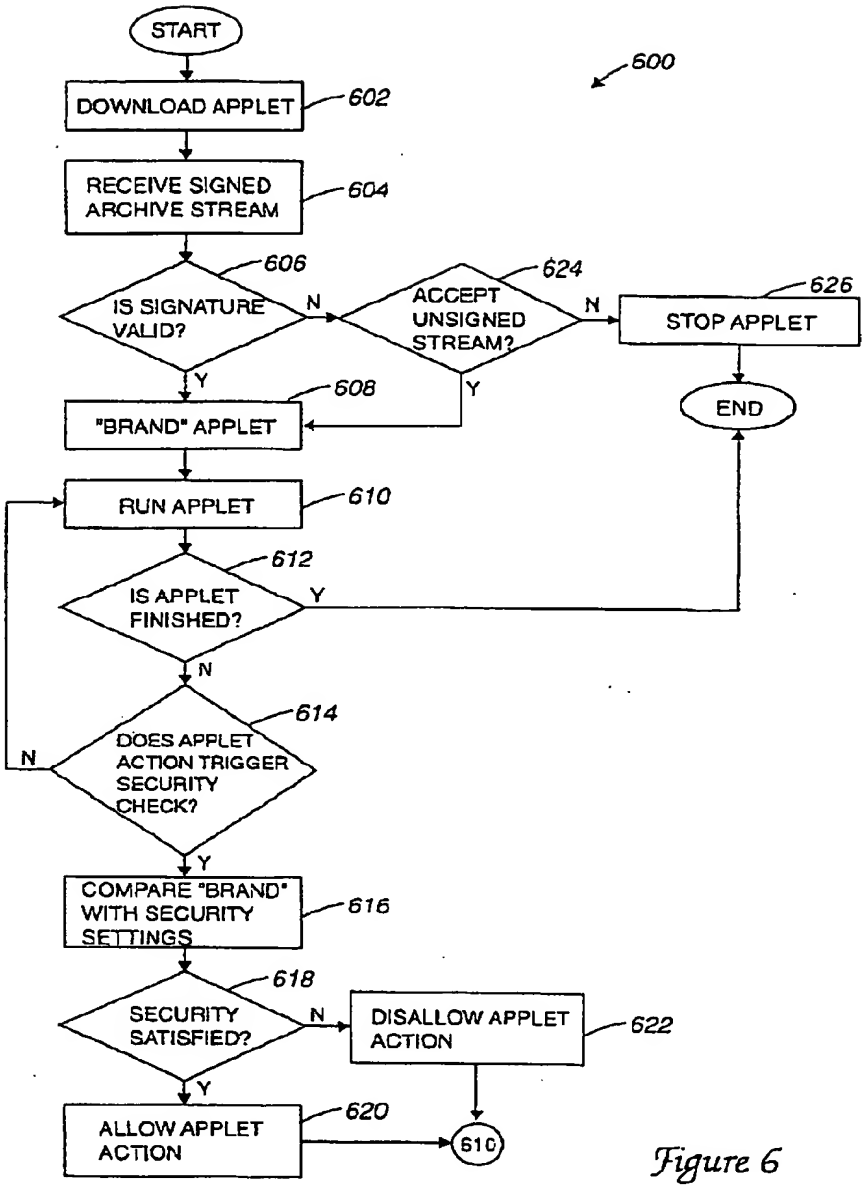

START
600
DOWNLOAD APPLET
602
RECEIVE SIGNED ARCHIVE STREAM
604
606
IS SIGNATURE VALID?
N
624
ACCEPT UNSIGNED STREAM?
N
626
STOP APPLET
Y
Y
END
608
"BRAND" APPLET
RUN APPLET
610
612
IS APPLET FINISHED?
Y
N
614
DOES APPLET ACTION TRIGGER SECURITY CHECK?
N
Y
COMPARE "BRAND" WITH SECURITY SETTINGS
616
618
SECURITY SATISFIED?
N
DISALLOW APPLET ACTION
622
Y
ALLOW APPLET ACTION
620
610

Figure 6

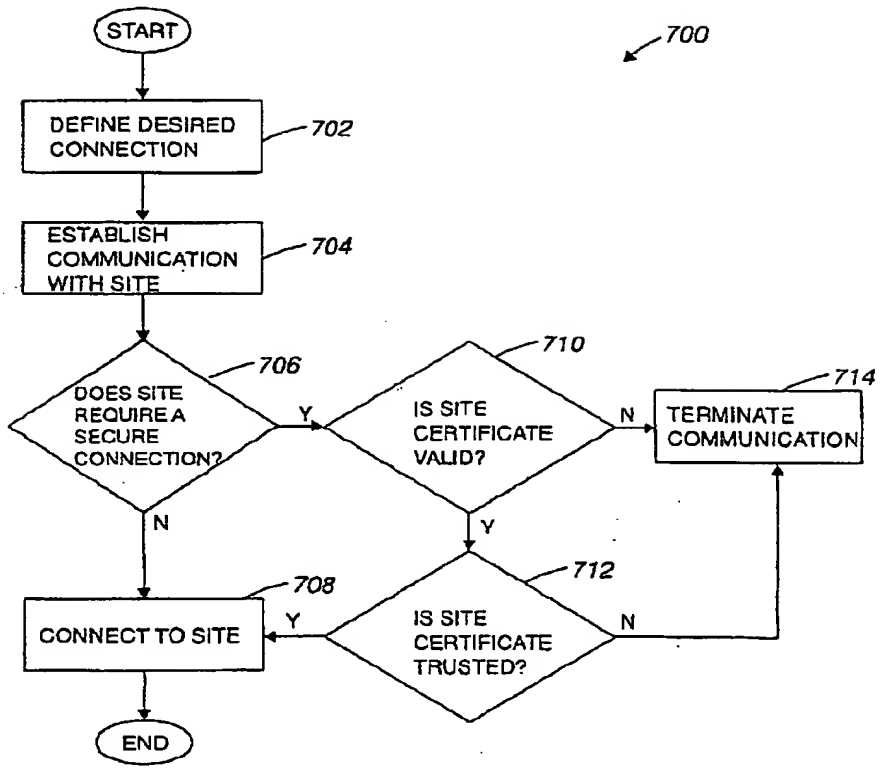

START
700
DEFINE DESIRED CONNECTION
702
ESTABLISH COMMUNICATION WITH SITE
704
DOES SITE REQUIRE A SECURE CONNECTION?
706
Y
N
IS SITE CERTIFICATE VALID?
710
N
Y
TERMINATE COMMUNICATION
714
IS SITE CERTIFICATE TRUSTED?
712
N
Y
CONNECT TO SITE
708
END

Figure 7

1. Abstract

ABSTRACT OF THE DISCLOSURE

Methods, apparatuses and products are provided for establishing and verifying the authenticity of data within one or more data files. In accordance with one aspect of the present invention, a method for verifying the authenticity of data involves providing at least one data file which includes an identifier and a signature file which includes the identifier for the data file as well as a digital signature. The digital signature is then verified using a computer system, and the identifier in the data file is compared with the identifier in the signature file using the computer system. In one embodiment, the identifier for the data file includes at least one certificate authority, site certificate, software publisher identifier, or a site name, and verifying the authenticity of data involves setting a security level for at least one of the certificate authority, said site certificate, said software publisher identifier, and said site name.

2. Representative Drawing

Fig. 4